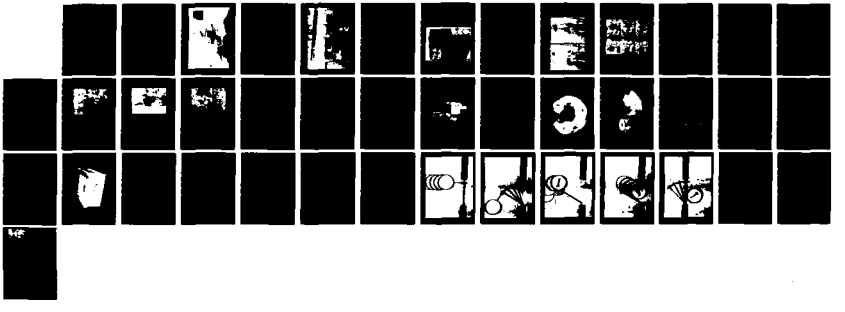
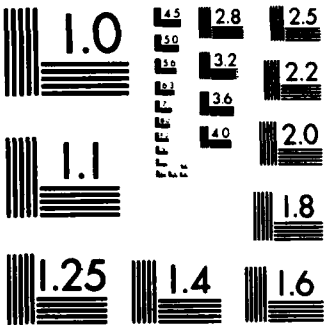
AD-A133 207 THE STANFORD CART AND THE CMU ROVER(U) CARNEGIE-MELLON 1/1
UNIV PITTSBURGH PA ROBOTICS INST H P MORAVEC 24 FEB 83
N00014-81-K-0503

UNCLASSIFIED F/G 6/4 NL





The Stanford Cart and The CMU Rover

Hans P. Moravec

The Robotics Institute
Carnegie-Mellon University
Pittsburgh, PA 15213

24 February 1983

The Stanford Cart work, conducted at the Stanford University Artificial Intelligence Laboratory, was supported over the years 1973 through 1980 by the Defense Advanced Research Projects Agency, the National Science Foundation and the National Aeronautics and Space Administration. The CMU Rover has been supported at the Carnegie-Mellon University Robotics Institute since 1981 by the Office of Naval Research under contract number N00014-81-K-0503.

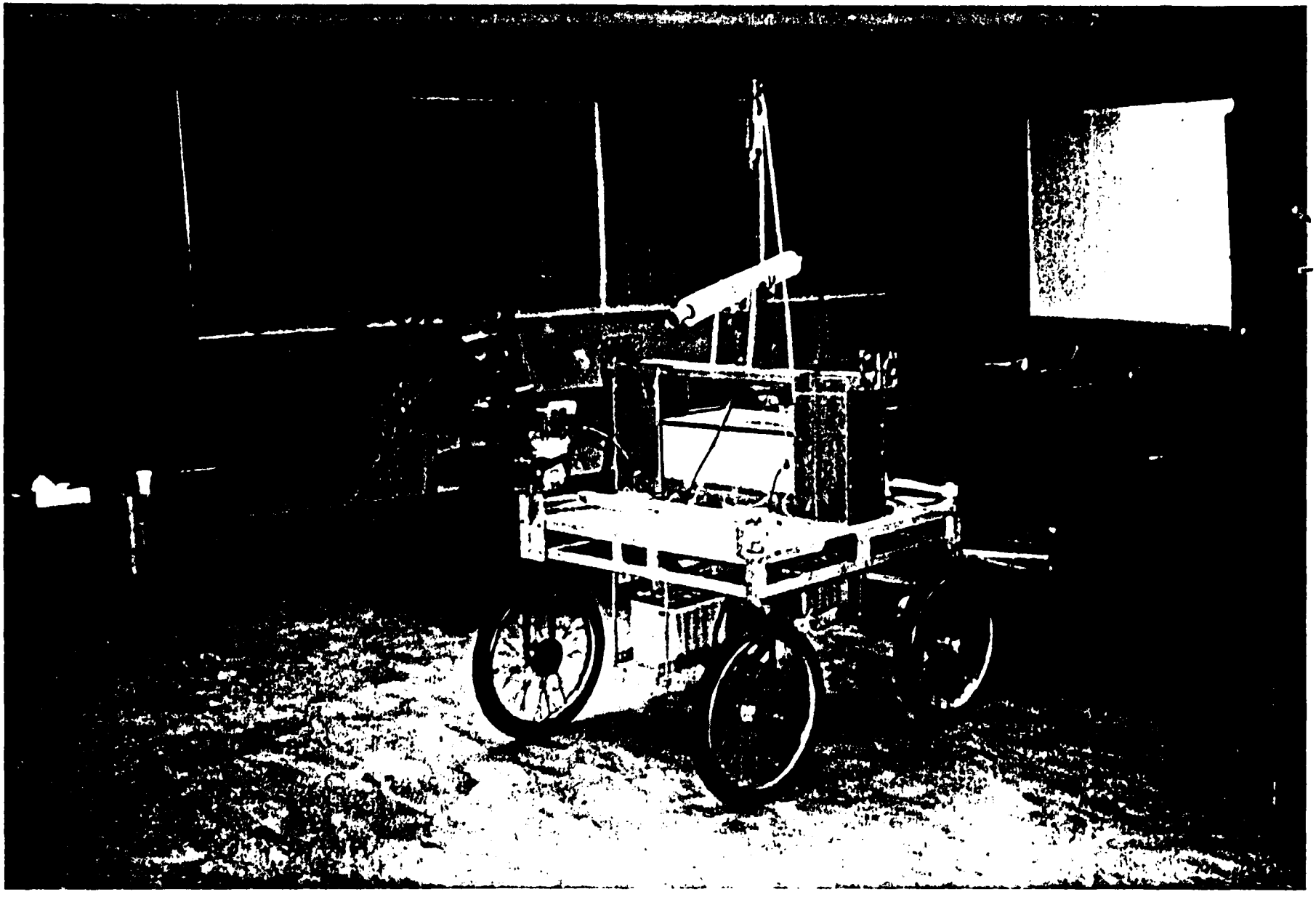


Figure 1: The Stanford Cart

Abstract

The Stanford Cart was a remotely controlled TV equipped mobile robot. A computer program was written which drove the Cart through cluttered spaces, gaining its knowledge of the world entirely from images broadcast by an onboard TV system. The CMU Rover is a more capable, and nearly operational, robot being built to develop and extend the Stanford work and to explore new directions.

The Cart used several kinds of stereopsis to locate objects around it in 3D and to deduce its own motion. It planned an obstacle avoiding path to a desired destination on the basis of a model built with this information. The plan changed as the Cart perceived new obstacles on its journey.

The system was reliable for short runs, but slow. The Cart moved one meter every ten to fifteen minutes, in lurches. After rolling a meter it stopped, took some pictures and thought about them for a long time. Then it planned a new path, executed a little of it, and paused again. It successfully drove the Cart through several 20 meter courses (each taking about five hours) complex enough to necessitate three or four avoiding swerves; it failed in other trials in revealing ways.

The Rover system has been designed with maximum mechanical and control system flexibility to support a wide range of research in perception and control. It features an omnidirectional steering system, a dozen onboard processors for essential real time tasks, and a large remote computer to be helped by a high speed digitizing/data playback unit and a high performance array processor. Distributed high level control software similar in organization to the Hearsay II speech understanding system and the beginnings of a vision library are being readied.

By analogy with the evolution of natural intelligence, we believe that incrementally solving the control and perception problems of an autonomous mobile mechanism is one of the best ways of arriving at general artificial intelligence.

Introduction

Experience with the Stanford Cart [8, 9, 11], a minimal computer controlled mobile camera platform, suggested to me that, while maintaining such a complex piece of hardware was a demanding task, the effort could be worthwhile from the point of view of artificial intelligence and computer vision research. A roving robot is a source of copious and varying visual and other sensory data which force the development of general techniques if the controlling programs are to be even minimally successful. By contrast the (also important) work with disembodied data and fixed robot systems often focuses on relatively restricted stimulii and small image sets, and improvements tend to be in the direction of specialization. Drawing an analogy with the natural world, I believe it is no mere co-incidence that in all cases imaging eyes and large brains evolved in animals that first developed high mobility.

The Stanford Cart

The Cart [10] was a minimal remotely controlled TV equipped mobile robot (Figure 1) which lived at the Stanford Artificial Intelligence Laboratory (SAIL). A computer program was written which drove the Cart through cluttered spaces, gaining its knowledge of the world entirely from images broadcast by the onboard TV system (Figure 2).

The Cart used several kinds of stereo vision to locate objects around it in 3D and to deduce its own motion. It planned an obstacle avoiding path to a desired destination on the basis of a model built with this information. The plan changed as the Cart perceived new obstacles on its journey.

The system was reliable for short runs, but slow. The Cart moved one meter every ten to fifteen minutes,

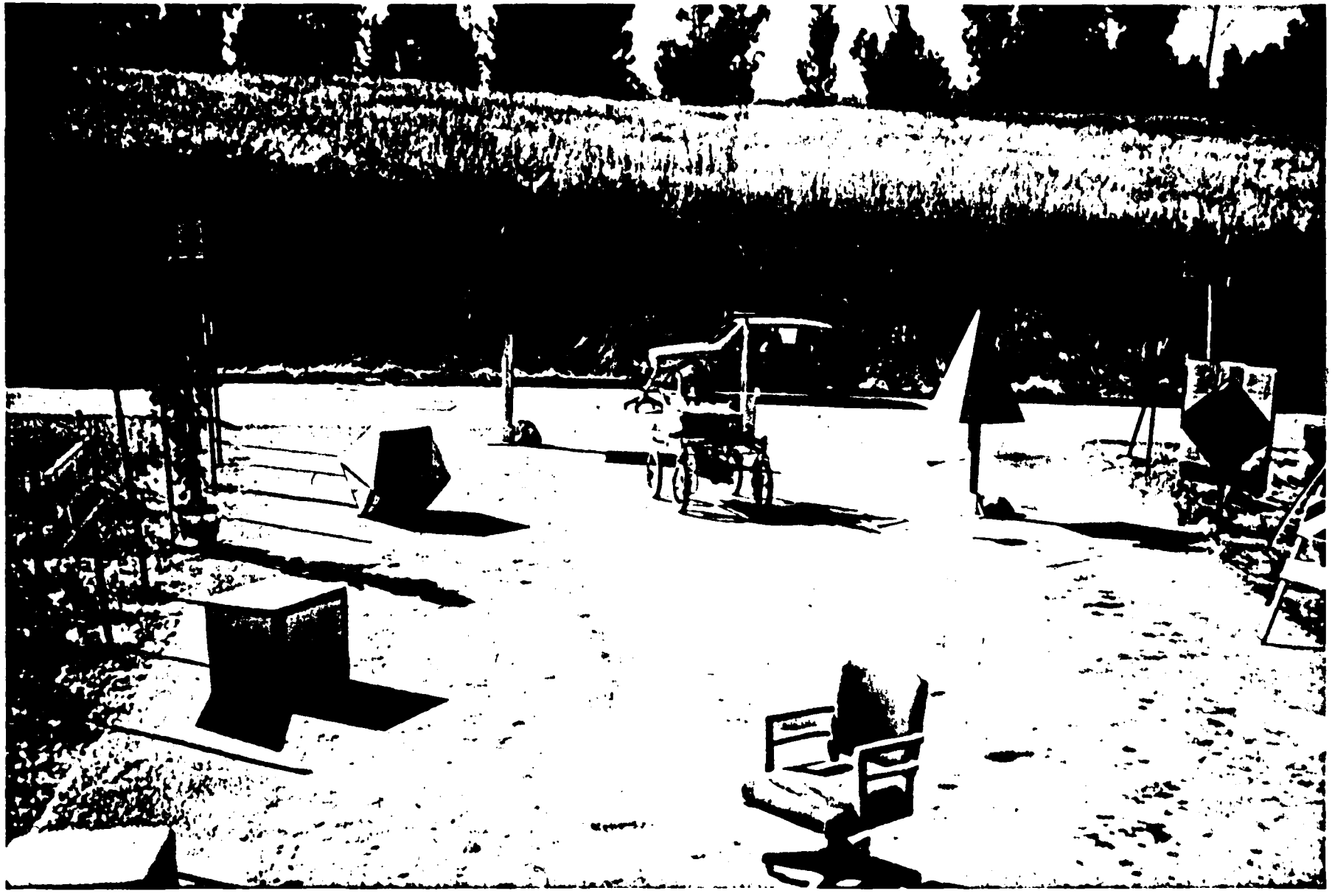


Figure 2: The Cart on an Obstacle Course

in lurches. After rolling a meter it stopped, took some pictures and thought about them for a long time. Then it planned a new path, executed a little of it, and paused again.

It successfully drove the Cart through several 20 meter courses (each taking about five hours) complex enough to necessitate three or four avoiding swerves. Some weaknesses and possible improvements were suggested by these and other, less successful, runs.

A Cart Run

A run began with a calibration of the Cart's camera. The Cart was parked in a standard position in front of a wall of carefully painted spots. A calibration program noted the disparity in position of the spots in the image seen by the camera with their position predicted from an idealized model of the situation. It calculated a distortion correction polynomial which related these positions, and which was used in subsequent ranging calculations (Figure 3).

The Cart was then manually driven to its obstacle course (littered with large and small debris) and the obstacle avoiding program was started. It began by asking for the Cart's destination, relative to its current position and heading. After being told, say, 50 meters forward and 20 to the right, it began its maneuvers. It activated a mechanism which moved the TV camera, and digitized nine pictures as the camera slid in precise steps from one side to the other along a 50 cm track.

A subroutine called the *interest operator* was applied to the one of these pictures. It picked out 30 or so particularly distinctive regions (features) in this picture. Another routine called the *correlator* looked for these same regions in the other frames (Figure 4). A program called the *camera solver* determined the three dimensional position of the features with respect to the Cart from their apparent movement from image to image (Figure 5).

The *navigator* planned a path to the destination which avoided all the perceived features by a large safety margin. The program then sent steering and drive commands to the Cart to move it about a meter along the planned path. The Cart's response to such commands was not very precise. The camera was then operated as before, and nine new images were acquired. The control program used a version of the correlator to find as many of the features from the previous location as possible in the new pictures, and applied the camera solver. The program then deduced the Cart's actual motion during the step from the apparent three dimensional shift of these features. Some of the features were pruned during this process, and the interest operator was invoked to add new ones.

This repeated until the Cart arrived at its destination or until some disaster terminated the program. Figures 6 and 7 document the Cart's internal world model at two points during a sample run.

Some Details

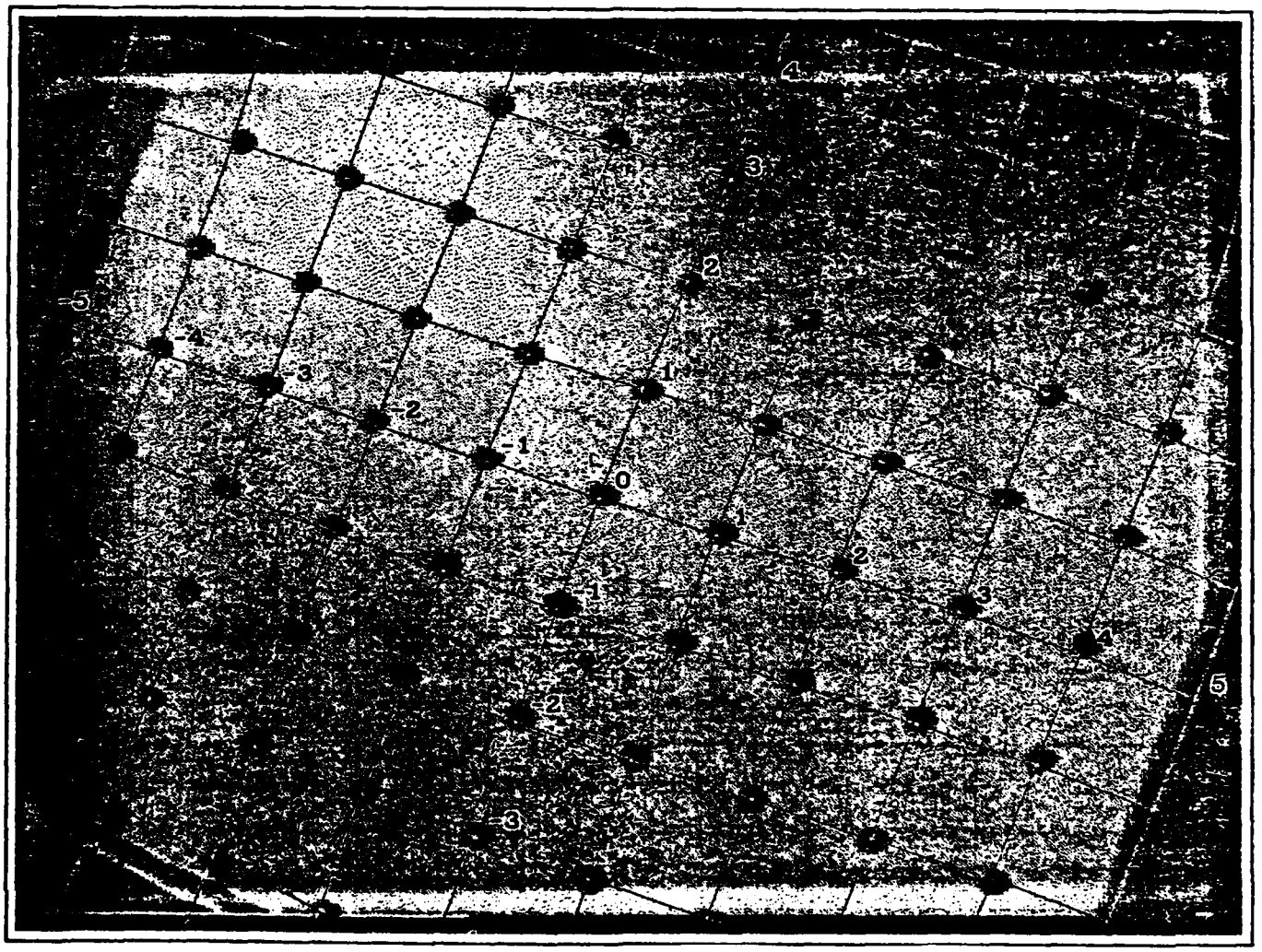
The Cart's vision code made extensive use of a reductions of each acquired image. Every digitized image was stored as the original picture accompanied by a pyramid of smaller versions of the image reduced in linear size by powers of two, each successive reduction obtained from the last by averaging four pixels into one.

Camera Calibration

The camera's focal length and geometric distortion were determined by parking the Cart a precise distance in front of a wall of many spots and one cross. A program digitized an image of the spot array, located the spots and the cross, and constructed a two dimensional polynomial that related the position of the spots in the image to their position in an ideal unity focal length camera, and another polynomial that converted points

Figure 3: A Cart's Eye View of the Calibration Grid

The Cart camera's focal length and distortions were determined by parking the Cart a precise distance in front of, and aiming its camera at, a carefully painted array of spots pattern, and running a calibration program. The program located the spots in the image and fitted a 2 dimensional, 3rd degree polynomial which converted actual positions in the image to co-ordinates in an ideal unity focal length camera. The picture presented here was obtained by running a corresponding grid in the unity focal length frame through the inverse function of the polynomial so obtained and superimposing it on the raw spot image.



from the ideal zcamera to points in the image. These polynomials were used to correct the positions of perceived objects in later scenes (Figure 3).

The algorithm began by determining the array's approximate spacing and orientation. It reduced by averaging and trimmed the picture to 64 by 64, calculated the Fourier transform of the reduced image and took its power spectrum, arriving at a 2D transform symmetric about the origin, and having strong peaks at frequencies corresponding to the horizontal and vertical and half-diagonal spacings, with weaker peaks at the harmonics. It multiplied each point $[i,j]$ in this transform by point $[-j,i]$ and points $[j-i,j+i]$ and $[i+j,j-i]$, effectively folding the primary peaks onto one another. The strongest peak in the 90 degree wedge around the y axis gave the spacing and orientation information needed by the next part of the process.

The interest operator described later was applied to roughly locate a spot near the center of the image. A special operator examined a window surrounding this position, generated a histogram of intensity values within the window, decided a threshold for separating the black spot from the white background, and calculated the centroid and first and second moment of the spot. This operator was again applied at a displacement from the first centroid indicated by the orientation and spacing of the grid, and so on, the region of found spots growing outward from the seed.

A binary template for the expected appearance of the cross in the middle of the array was constructed from the orientation/spacing data from the Fourier transform. The area around each of the found spots was thresholded on the basis of the expected cross area, and the resulting two valued pattern was convolved with the cross template. The closest match in the central portion of the picture was declared to be the origin.

Two least-squares polynomials (one for X and one for Y) of third (or sometimes fourth) degree in two variables, relating the actual positions of the spots to the ideal positions in a unity focal length camera, were then generated and written into a file. The polynomials were used in the obstacle avoider to correct for camera roll, tilt, lens focal length and long term variations in the vidicon geometry.

Interest Operator

The Cart vision code dealt with localized image patches called features. A feature is conceptually a point in the three dimensional world, but it was found by examining localities larger than points in pictures. A feature was good if it could be located unambiguously in different views of a scene. A uniformly colored region or a simple edge is not good because its parts are indistinguishable. Regions, such as corners, with high contrast in orthogonal directions are best.

New features in images were picked by a subroutine called the *interest operator*, which returned regions that were local maxima of a directional variance measure, defined below. The idea was to select a relatively uniform scattering of good features over the image, so that a few would likely be picked on every visible object while textureless areas and simple edges were avoided.

Directional variance was measured over small square windows. Sums of squares of differences of pixels adjacent in each of four directions (horizontal, vertical and two diagonals) over each window were calculated, and the window's interest measure was the minimum of these four sums. Features were chosen where the interest measure had local maxima. The chosen features were stored in an array, sorted in order of decreasing interest measure (Figure 4 top).

Once a feature was chosen, its appearance was recorded as series of excerpts from the reduced image sequence. A 6 by 6 window was excised around the feature's location from each of the variously reduced pictures. Only a tiny fraction of the area of the original (unreduced) image was extracted. Four times as much area (but the same number of pixels) of the x2 reduced image was stored, sixteen times as much of the x4 reduction, and so on until at some level we had the whole image. The final result was a series of 6 by 6 pictures, beginning with a very blurry rendition of the whole picture, gradually zooming in linear expansions

Figure 4: Interest Operator and Correlator Results

The upper picture shows points picked out by an application of the Interest Operator. The lower picture shows the Correlator's attempt to find the same points in an image of the same scene taken from a different point of view.

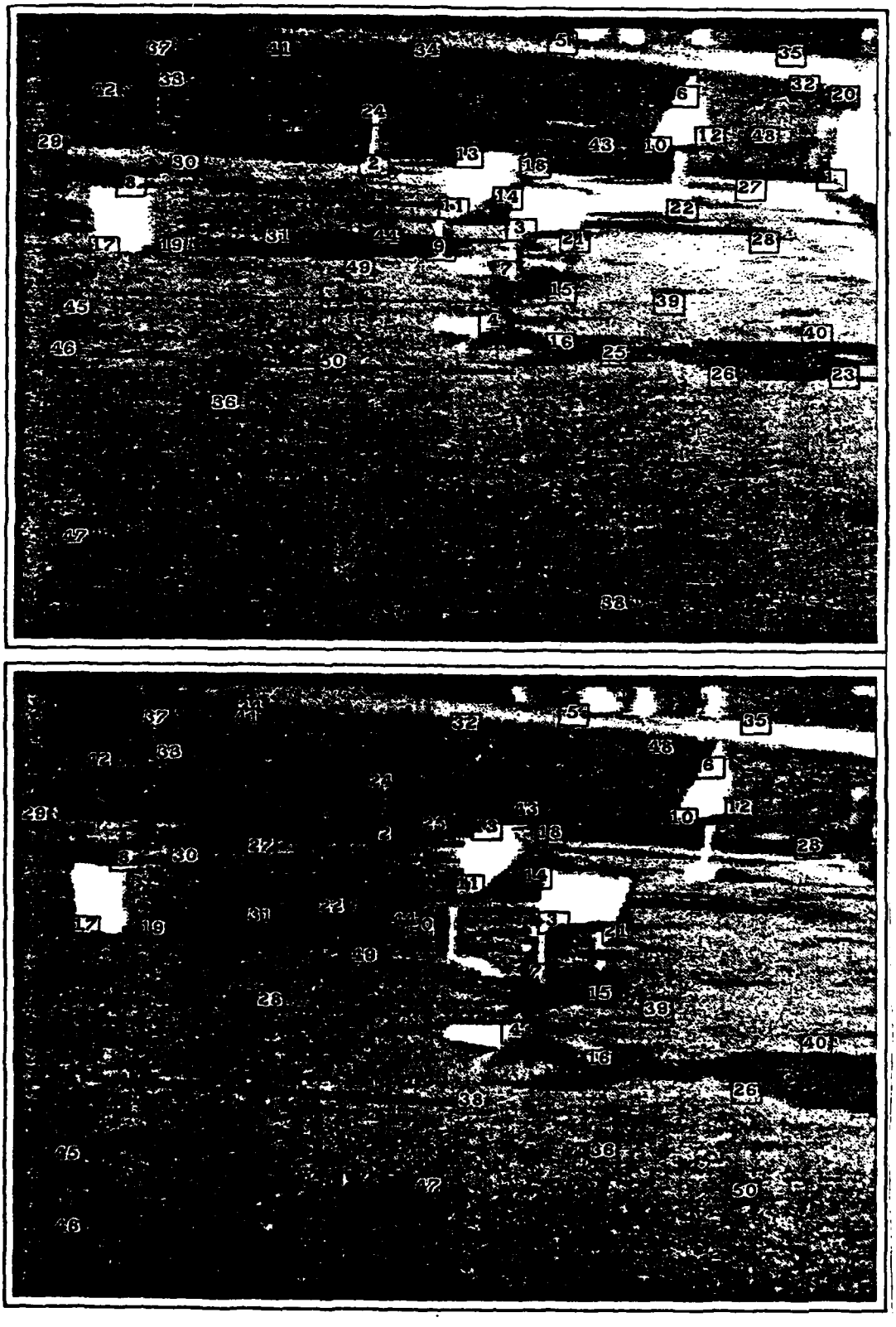
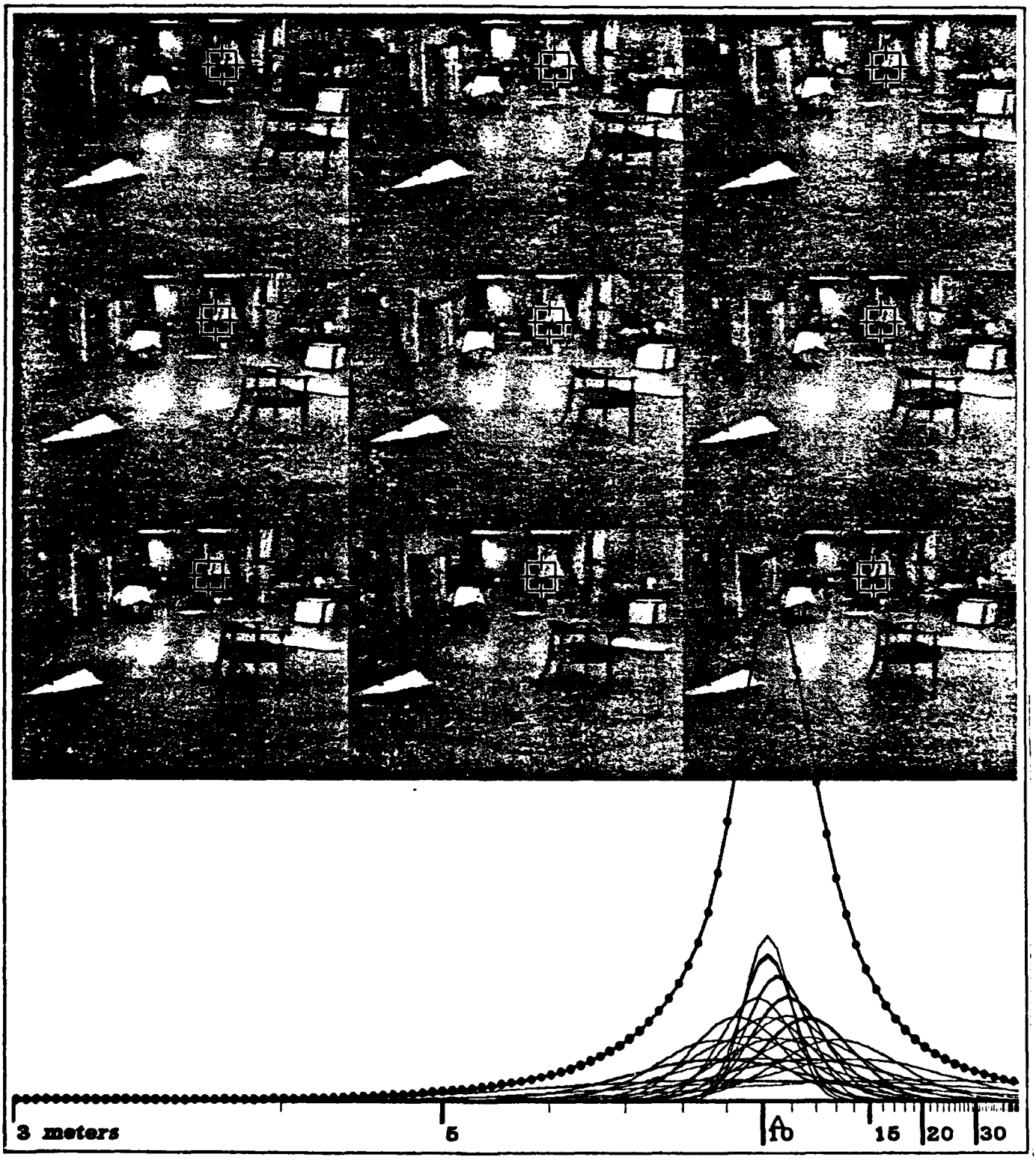


Figure 5: Slider Stereo

A typical ranging. The nine pictures are from a slider scan. The interest operator chose the marked feature in the central image, and the correlator found it in the other eight. The small curves at bottom are distance measurements of the feature made from pairs of the images. The large beaded curve is the sum of the measurements over all 36 pairings. The horizontal scale is linear in inverse distance.



of two to a sharp closeup of the feature.

Correlation

Deducing the 3D location of features from their projections in 2D images requires that we know their position in two or more such images. The *correlator* was a subroutine that, given a feature description produced by the interest operator from one image, found the best match in a different, but similar, image. Its search area could be the entire new picture, or a rectangular sub-window.

The search used a coarse to fine strategy that began in reduced versions of the pictures. Typically the first step took place at the x16 (linear) reduction level. The 6 by 6 window at that level in the feature description, that covered about one seventh of the total area of the original picture, was convolved with the search area in the correspondingly reduced version of the second picture. The 6 by 6 description patch was moved pixel by pixel over the approximately 15 by 16 destination picture, and a correlation coefficient was calculated for each trial position. The position with the best match was recorded. The 6x6 area it occupied in the second picture was mapped to the x8 reduction level, where the corresponding region was 12 pixels by 12. The 6 by 6 window in the x8 reduced level of the feature description was then convolved with this 12 by 12 area, and the position of best match was recorded and used as a search area for the x4 level. The process continued, matching smaller and smaller, but more and more detailed windows until a 6 by 6 area was selected in the unreduced picture (Figure 4 bottom).

This "divide and conquer" strategy was in general able to search an entire picture for a match more quickly (because most of the searching was done at high reduction levels) and more reliably (because context up to and including the entire picture guided the search) than a straightforward convolution over even a very restricted search area.

Slider Stereo

At each pause on its computer controlled itinerary the Cart slid its camera from left to right on a 52 cm track, taking 9 pictures at precise 6.5 cm intervals. Points were chosen in the fifth (middle) of these 9 images, either by the correlator to match features from previous positions, or by the interest operator. The camera slid parallel to the horizontal axis of the (distortion corrected) camera co-ordinate system, so the parallax-induced displacement of features in the 9 pictures was purely horizontal.

The correlator was applied eight times to look for the points chosen in the central image in each of other eight pictures. The search was restricted to a narrow horizontal band. This had little effect on the computation time, but it reduced the probability of incorrect matches. In the case of correct matches, the distance to the feature was inversely proportional to its displacement from one image to another. The uncertainty in such a measurement is the difference in distance a shift one pixel in the image would make. The uncertainty varies inversely with the physical separation of the camera positions where the pictures were taken (the stereo baseline). Long baselines give more accurate distance measurements.

After the correlation step the program knew a feature's position in nine images. It considered each of the 36 (9 values taken 2 at a time) possible image pairings as a stereo baseline, and recorded the estimated (inverse) distance of the feature in a histogram. Each measurement added a little normal curve to the histogram, with mean at the estimated distance, and standard deviation inversely proportional to the baseline, reflecting the uncertainty. The area under each curve was made proportional to the product of the goodness of the matches in the two images (in the central image this quantity is taken as unity), reflecting the confidence that the correlations were correct. The distance to the feature was indicated by the largest peak in the resulting histogram, if this peak was above a certain threshold. If below, the feature was forgotten (Figure 5).

The correlator sometimes matched features incorrectly. The distance measurements from incorrect

matches in different pictures were not consistent. When the normal curves from 36 pictures pairs are added up, the correct matches agree with each other, and build up a large peak in the histogram, while incorrect matches spread themselves more thinly. Two or three correct correlations out of the eight usually built a peak sufficient to offset a larger number of errors. In this way eight applications of a mildly reliable operator interacted to make a very reliable distance measurement.

Motion Stereo

After having determined the 3D location of objects at one position, the computer drove the Cart about a meter forward. At the new position it slid the camera and took nine pictures. The correlator was applied in an attempt to find all the features successfully located at the previous position. Feature descriptions extracted from the central image at the last position were searched for in the central image at the new stopping place.

Slider stereo then determined the distance of the features so found from the Cart's new position. The program now knew the 3D position of the features relative to its camera at the old and the new locations. Its own movement was deduced from 3D co-ordinate transform that related the two.

The program first eliminated mis-matches in the correlations between the central images at two positions. Although it didn't yet have the co-ordinate transform between the old and new camera systems, the program knew the distance between pairs of feature positions should be the same in both. It made a matrix in which element $[i,j]$ is the absolute value of the difference in distances between points i and j in the first and second co-ordinate systems divided by the expected error (based on the one pixel uncertainty of the ranging). Each row of this matrix was summed, giving an indication of how much each point disagreed with the other points. The idea is that while points in error disagree with virtually all points, correct positions agree with all the other correct ones, and disagree only with the bad ones. The worst point was deleted, and its effect removed from the remaining points in the row sums. This pruning was repeated until the worst error was within the error expected from the ranging uncertainty.

After the pruning, the program had a number of points, typically 10 to 20, whose position error was small and pretty well known. The program trusted these, and recorded them in its world model, unless it had already done so at a previous position. The pruned points were forgotten forevermore.

The 3D rotation and translation that related the old and new Cart position was then calculated by a Newton's method iteration that minimized the sum of the squares of the distances between the transformed first co-ordinates and the raw co-ordinates of the corresponding points at the second position, with each term divided by the square of the expected uncertainty in the 3D position of the points involved.

Path Planning

The Cart vision system modelled objects as simple clouds of features. If enough features are found on each nearby object, this model is adequate for planning a non-colliding path to a destination. The features in the Cart's 3D world model can be thought of as fuzzy ellipsoids, whose dimensions reflect the program's uncertainty of their position. Repeated applications of the interest operator as the Cart moves caused virtually all visible objects to be become modelled as clusters of overlapping ellipsoids.

To simplify the problem, the ellipsoids were approximated by spheres. Those spheres sufficiently above the floor and below the Cart's maximum height were projected on the floor as circles. The one meter square Cart itself was modelled as a 3 meter circle. The path finding problem then became one of maneuvering the Cart's 3 meter circle between the (usually smaller) circles of the potential obstacles to a desired location. It is convenient (and equivalent) to conceptually shrink the Cart to a point, and add its radius to each and every obstacle. An optimum path in this environment will consist of either a straight run between start and finish, or a series of tangential segments between the circles and contacting arcs (imagine loosely laying a string from

start to finish between the circles, then pulling it tight).

The program converted the problem to a shortest path in graph search. There are four possible paths between each pair of obstacles because each tangent can approach clockwise or counterclockwise. Each tangent point became a vertex in the graph, and the distance matrix of the graph (which had an entry for each vertex pair) contained sums of tangential and arc paths, with infinities for blocked or impossible routes. The shortest distance in this space can be found with an algorithm whose running time is $O(n^3)$ in the number of vertices, and the Cart program was occasionally run using this exact procedure. It was run more often with a faster approximation that made each obstacle into only two vertices (one for each direction of circumnavigation).

A few other considerations were essential in path planning. The charted routes consisted of straight lines connected by tangent arcs, and were thus plausible paths for the Cart, which steered like an automobile. This plausibility was not necessarily true of the start of the planned route, which, as presented thus far, did not take the initial heading of the Cart into account. The plan could, for instance, include an initial segment going off 90 degrees from the direction in which the Cart pointed, and thus be impossible to execute. This was handled by including a pair of "phantom" obstacles along with the real perceived ones. The phantom obstacles had a radius equal to the Cart's minimum steering radius, and were placed, in the planning process, on either side of the Cart at such a distance that after their radius was augmented by the Cart's radius (as happened for all the obstacles), they just touched the Cart's centroid, and each other, with their common tangents being parallel to the direction of the Cart's heading. They effectively blocked the area made inaccessible to the Cart by its maneuverability limitations (Figure 6).

[7] describes an independently developed, but very similar, approach to finding paths around polygonal obstacles.

Path Execution

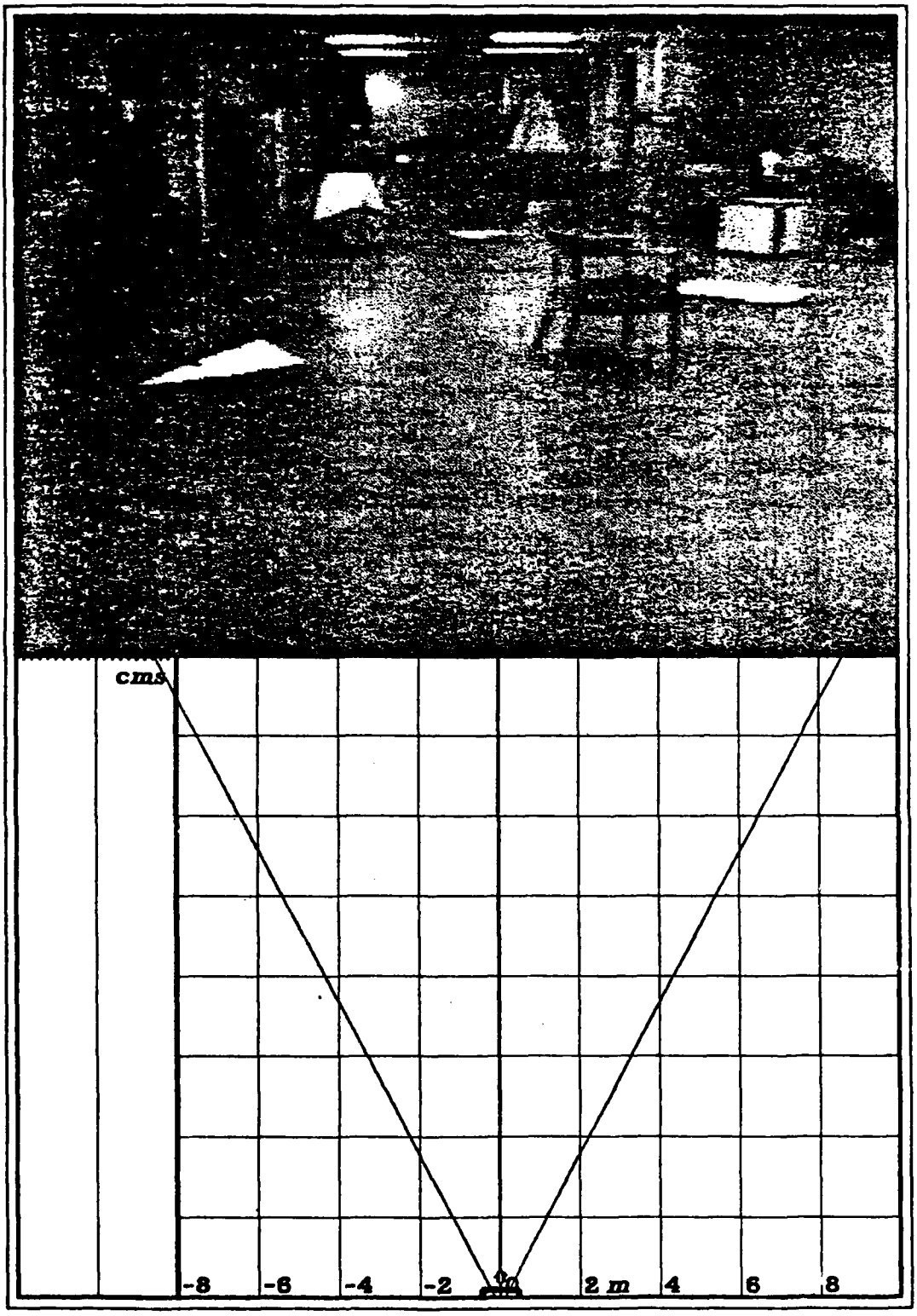
After the path to the destination had been chosen, a portion of it had to be implemented as steering and motor commands and transmitted to the Cart. The control system was primitive. The drive motor and steering motors could be turned on and off at any time, but there existed no means to accurately determine just how fast or how far they had gone. The program made the best of this bad situation by incorporating a model of the Cart that mimiced, as accurately as possible, the Cart's actual behavior. Under good conditions, as accurately as possible means about 20%; the Cart was not very repeatable, and was affected by ground slope and texture, battery voltage, and other less obvious externals.

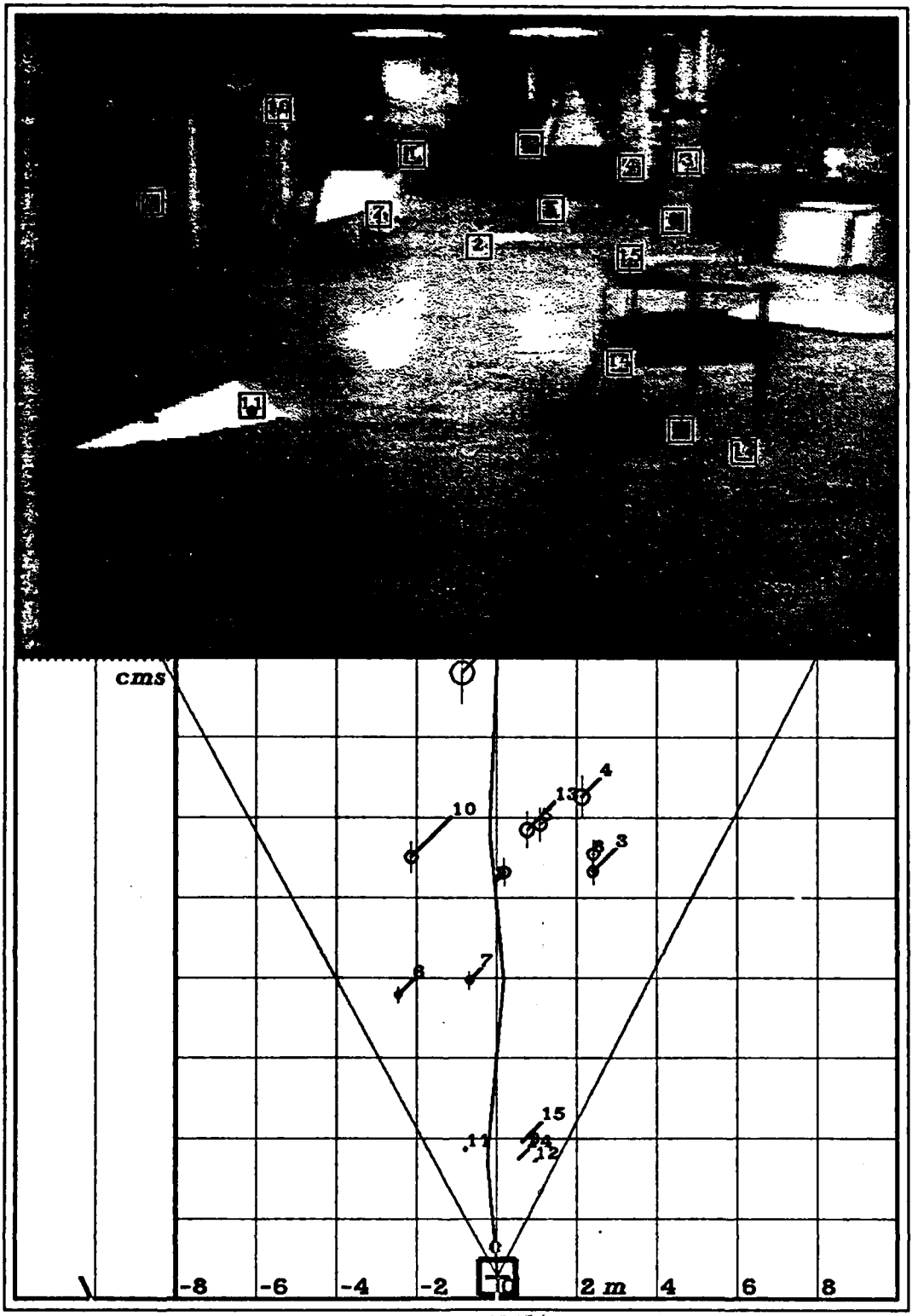
The path executing routine began by excising the first .75 meters of the planned path. This distance was chosen as a compromise between average Cart velocity, and continuity between picture sets. If the Cart moved too far between picture digitizing sessions, the picture would change too much for reliable correlations. This is especially true if the Cart turns (steers) as it moves. The image seen by the camera then pans across the field of view. The Cart had a wide angle lens that covers 60 degrees horizontally. The .75 meters, combined with the turning radius limit (5 meters) of the Cart resulted in a maximum shift in the field of view of 15 degrees, one quarter of the entire image.

The program examined the Cart's position and orientation at the end of the desired .75 meter lurch, relative to the starting position and orientation. The displacement was characterized by three parameters; displacement forward, displacement to the right and change in heading. In closed form the program computed a path that accomplished this movement in two arcs of equal radius, but different lengths. The resulting trajectory had a general "S" shape. Rough motor timings were derived from these parameters. The program then used a simulation that took into account steering and drive motor response to iteratively refine the solution.

Figure 6: A Cart Obstacle Run

This and the following diagram are plan views of the Cart's internal world model during a run of the obstacle avoiding program. The grid cells are two meter squares, conceptually on the floor. The Cart's own position is indicated by the small heavy square, and by the graph, indicating height, calibrated in centimeters, to the left of grid. Since the Cart never actually leaves or penetrates the floor, this graph provides an indication of the overall accuracy. The irregular, tick marked, line behind the Cart's position is the past itinerary of the Cart as deduced by the program. Each tick mark represents a stopping place. The picture at top of the diagrams is the view seen by the TV camera. The two rays projecting forward from the Cart position show the horizontal boundaries of the camera's field of view (as deduced by the camera calibration program). The numbered circles in the plan view are features located and tracked by the program. The centers of the circles are the vertical projections of the feature positions onto the ground. The size of each circle is the uncertainty (caused by finite camera resolution) in the feature's position. The length of the 45 degree line projecting to the upper right, and terminated by an identifying number, is the height of the feature above the ground, to the same scale as the floor grid. The features are also marked in the camera view, as numbered boxes. The thin line projecting from each box to a lower blob is a stalk which just reaches the ground, in the spirit of the 45 degree lines in the plan view. The irregular line radiating forwards from the Cart is the planned future path. This changes from stop to stop, as the Cart fails to obey instructions properly, and as new obstacles are detected. The small ellipse a short distance ahead of the Cart along the planned path is the planned position of the next stop.


cms
-8
-6
-4
-2
2 m
4
6
8


cms
10
13
4
3
8
7
15
11
12
-8
-6
-4
-2
0
2 m
4
6
8

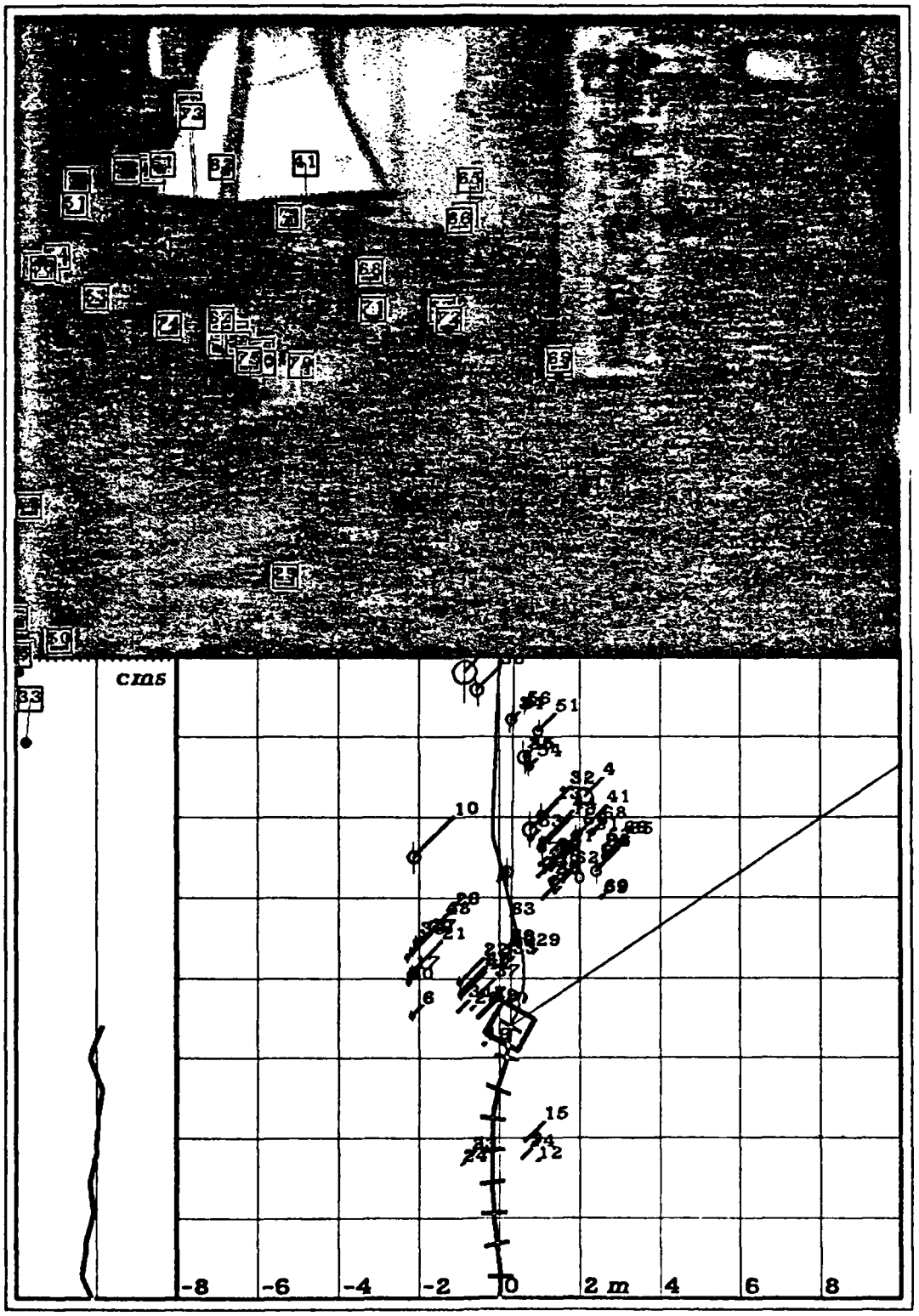


Figure 7:

After the 11'th lurch the Cart has rounded the chair, the icosahedron and is working on the cardboard tree. The world model has suffered some accumulated drift error, and the oldest acquired features are considerably misplaced.

Cart Experiments

The system described above only incompletely fulfills some of the hopes I had when the work began many years ago.

One of the most serious limitations was the excruciating slowness of the program. In spite of my best efforts, and many compromises, in the interest of speed, it took 10 to 15 minutes of real time to acquire and consider the images at each meter long lurch, on a lightly loaded DEC KL-10. This translated to an effective Cart velocity of 3 to 5 meters an hour. Interesting obstacle courses (two or three major obstacles, spaced far enough apart to permit passage within the limits of the Cart's size and maneuverability) were about 20 meters long, so interesting Cart runs took 5 hours.

The reliability of individual moves was high, as it had to be for a twenty lurch sequence to have any chance of succeeding, but the demanding nature of each full run and the limited amount of time available for testing (discussion of which is beyond the scope of this paper) after the bulk of the program was debugged, ensured that many potential improvements were left untried. Three full (about 20 meter) runs were digitally recorded and filmed, two indoors and one outdoors. Two indoor false starts, aborted by failure of the program to perceive an obstacle, were also recorded. The two long indoor runs were nearly perfect.

In the first long indoor run, the Cart successfully slalomed its way around a chair, a large cardboard icosahedron, and a cardboard tree then, at a distance of about 16 meters, encountered a cluttered wall and backed up several times trying to find a way around it (Figures 6 and 7 are snapshots from this run).

The second long indoor run involved a more complicated set of obstacles, arranged primarily into two overlapping rows blocking the goal. I had set up the course hoping the Cart would take a long, picturesque (the runs were being filmed) S shaped path around the ends of the rows. To my chagrin, it instead tried for a tricky shortcut. The Cart backed up twice to negotiate the tight turn required to go around the first row, then executed several tedious steer forward / back up moves, lining itself up to go through a gap barely wide enough in the second row. This run had to be terminated, sadly, before the Cart had gone through the gap because of declining battery charge and increasing system load.

The outdoor run was less successful. It began well; in the first few moves the program correctly perceived a chair directly in front of the camera, and a number of more distant cardboard obstacles and sundry debris. Unfortunately, the program's idea of the Cart's own position became increasingly wrong. At almost every lurch, the position solver deduced a Cart motion considerably smaller than the actual move. By the time the Cart had rounded the foreground chair, its position model was so far off that the distant obstacles were replicated in different positions in the Cart's confused world model, because they had been seen early in the run and again later, to the point where the program thought an actually existing distant clear path was blocked. I restarted the program to clear out the world model when the planned path became too silly. At that time the Cart was four meters in front of a cardboard icosahedron, and its planned path lead straight through it. The newly re-incarnated program failed to notice the obstacle, and the Cart collided with it. I manually moved the icosahedron out of the way, and allowed the run to continue. It did so uneventfully, though there were continued occasional slight errors in the self position deductions. The Cart encountered a large cardboard tree towards the end of this journey and detected a portion of it only just in time to squeak by without colliding (Figure 2 was a photograph taken during this run).

The two short abortive indoor runs involved setups nearly identical to the two-row successful long run described one paragraph ago. The first row, about three meters in front of the Cart's starting position contained a chair, a real tree (a small cypress in a planting pot), and a polygonal cardboard tree. The Cart saw the chair instantly and the real tree after the second move, but failed to see the cardboard tree ever. Its planned path around the two obstacles it did see put it on a collision course with the unseen one. Placing a chair just ahead of the cardboard tree fixed the problem, and resulted in a successful run. The finished

program never had trouble with chairs.

Problems

These tests revealed some weaknesses in the program. The system did not see simple polygonal (bland and featureless) objects reliably, and its visual navigation was fragile under certain conditions. Examination of the program's internal workings suggested some causes and possible solutions.

The program sometimes failed to see obstacles lacking sufficient high contrast detail within their outlines. In this regard, the polygonal tree and rock obstacles I whimsically constructed to match diagrams from a 3D drawing program, were a terrible mistake. In none of the test runs did the programs ever fail to see a chair placed in front of the Cart, but half the time they did fail to see a pyramidal tree or an icosahedral rock made of clean white cardboard. These contrived obstacles were picked up reliably at a distance of 10 to 15 meters, silhouetted against a relatively unmoving (over slider travel and Cart lurches) background, but were only rarely and sparsely seen at closer range, when their outlines were confused by a rapidly shifting background, and their bland interiors provided no purchase for the interest operator or correlator. Even when the artificial obstacles were correctly perceived, it was by virtue of only two to four features. In contrast, the program usually tracked five to ten features on nearby chairs.

In the brightly sunlit outdoor run the artificial obstacles had another problem. Their white coloration turned out to be much brighter than any "naturally" occurring extended object. These super bright, glaring, surfaces severely taxed the very limited dynamic range of the Cart's vidicon/digitizer combination. When the icosahedron occupied 10% of the camera's field of view, the automatic target voltage circuit in the electronics turned down the gain to a point where the background behind the icosahedron appeared nearly solid black.

The second major problem exposed by the runs was glitches in the Cart's self-position model. This model was updated after a lurch by finding the 3D translation and rotation that best related the 3D position of the set of tracked features before and after the lurch. In spite of the extensive pruning that preceded this step, (and partly because of it, as is discussed later) small errors in the measured feature positions sometimes caused the solver to converge to the wrong transform, giving a position error beyond the expected uncertainty. Features placed into the world model before and after such a glitch were not in the correct relative positions. Often an object seen before was seen again after, now displaced, with the combination of old and new positions combining to block a path that was in actuality open.

This problem showed up mainly in the outdoor run. I had observed it indoors in the past, in simple mapping runs, before the entire obstacle avoider was assembled. There appear to be two major causes for it, and a wide range of supporting factors.

Poor seeing, resulting in too few correct correlations between the pictures before and after a lurch, was one culprit. The highly redundant nine eyed stereo ranging was very reliable, and caused few problems, but the non-redundant correlation necessary to relate the position of features before and after a lurch, was error prone. Sometimes the mutual-distance invariance pruning that followed was overly aggressive, and left too few points for a stable least-squares co-ordinate fit.

The outdoor runs encountered another problem. The program ran so slowly that shadows moved significantly (up to a half meter) between lurches. Their high contrast boundaries were favorite points for tracking, enhancing the program's confusion.

Quick Fixes

Though elaborate (and thus far untried in our context) methods such as edge matching may greatly improve the quality of automatic vision in future, subsequent experiments with the program revealed some modest incremental improvements that would have solved most of the problems in the test runs.

The issue of unseen cardboard obstacles turns out to be partly one of over-conservatism on the program's part. In all cases where the Cart collided with an obstacle it had correctly ranged a few features on the obstacle in the prior nine-eyed scan. The problem was that the much more fragile correlation between vehicle forward moves failed, and the points were rejected in the mutual distance test. Overall the nine-eyed stereo produced very few errors. If the path planning stage had used the pre-pruning features (still without incorporating them permanently into the world model) the runs would have proceeded much more smoothly. All of the most vexing false negatives, in which the program failed to spot a real obstacle, would have been eliminated. There would have been a very few false positives, in which non-existent ghost obstacles would have been perceived. One or two of these might have caused an unnecessary swerve or backup. But such ghosts would not pass the pruning stage, and the run would have recovered after the initial, non-catastrophic, glitch.

The self-position confusion problem is related, and in retrospect may be considered a trivial bug. When the path planner computed a route for the Cart, another subroutine took a portion of this plan and implemented it as a sequence of commands to be transmitted to the Cart's steering and drive motors. During this process it ran a simulation that modelled the Cart acceleration, rate of turning and so on, and which provided a prediction of the Cart's position after the move. With the old hardware the accuracy of this prediction was not great, but it nevertheless provided much a priori information about the Cart's new position. This information was used, appropriately weighted, in the least-squares co-ordinate system solver that deduced the Cart's movement from the apparent motion in 3D of tracked features. It was not used, however, in the mutual distance pruning step that preceeded this solving. When the majority of features had been correctly tracked, failure to use this information did not hurt the pruning. But when the seeing was poor, it could make the difference between choosing a spuriously agreeing set of mis-tracked features and the small correctly matched set. Incorporating the prediction into the pruning, by means of a heavily weighted point that the program treats like another tracked feature, removed almost all the positioning glitches when the program was fed the pictures from the outdoor run.

More detail on all these areas can be found in [10].

The CMU Rover

The major impediments to serious extensions of the Cart work were limits to available computation, resulting in debilitatingly long experimental times, and the very minimal nature of the robot hardware, which precluded inexpensive solutions for even most basic functions (like "roll a meter forward").

We are addressing these problems at CMU in an ambitious new effort centered around a new, small but sophisticated mobile robot dubbed the CMU Rover. The project so far has been focused on developing a smoothly functional and highly capable vehicle and associated support system which will serve a wide variety of future research.

The shape, size, steering arrangements and onboard and external processing capabilities of the Rover system were chosen to maximize the flexibility of the system (naturally limited by present day techniques).

The robot is cylindrical, about a meter tall and 55 cm in diameter (Figure 8) and has three individually steerable wheel assemblies which give it a full three degrees of freedom of mobility in the plane (Figures 9, 10). Initially it will carry a TV camera on a pan/tilt/slide mount, several short range infrared and long range sonar proximity detectors, and contact switches. Our design calls for about a dozen onboard processors (at

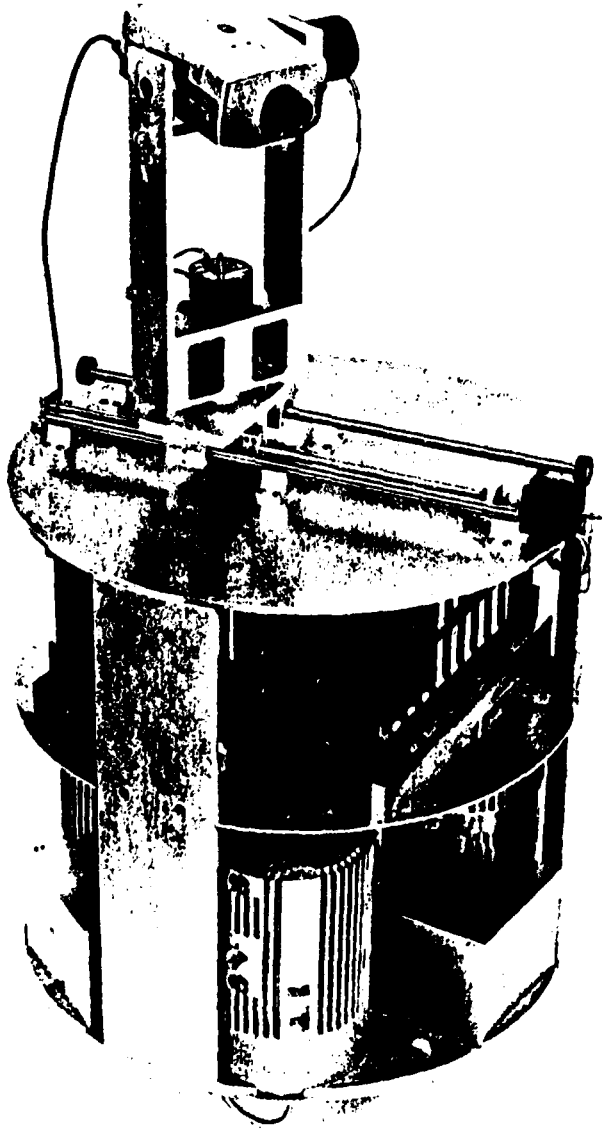


Figure 8: The CMU Rover

least half of them powerful 16 bit MC68000s) for high speed local decision making, servo control and communication (Figure 13).

Serious processing power, primarily for vision, is to be provided at the other end of a remote-control link by a combination of a host computer VAX 11/780 an ST-100 array processor (a new machine from a new company, Star Technologies Inc., which provides 100 million floating point operations per second) and a specially designed high performance analog data acquisition and generation device. The Stanford Cart used fifteen minutes of computer time to move a meter. With this new CMU hardware, and some improved algorithms, we hope to duplicate (and improve on) this performance in a system that runs at least ten times as fast, leaving room for future extensions.

The fifteen minutes for each meter-long move in the Cart's obstacle avoiding program came in three approximately equal chunks. The first five minutes were devoted to digitizing the nine pictures from a slider scan. Though the SAIL computer had a flash digitizer which sent its data through a disk channel into main memory at high speed, it was limited by a poor sync detector. Often the stored picture was missing scanlines, or had lost vertical sync, i.e. had rolled vertically. In addition, the image was quantized to only four bits per sample; 16 grey levels. To make one good picture the program digitized 30 raw images in rapid succession, intercompared them to find the largest subset of nearly alike pictures (on the theory that the non-spoiled ones would be similar, but the spoiled ones would differ even from each other) and averaged this "good" set to obtain a less noisy image with six bits per pixel. The new digitizing hardware, which can sample a raw analog waveform, and depends on software in the array processor to do sync detection, should cut the total time to under one second per picture. The next five minutes was spent doing the low level vision; reducing the images, sometimes filtering them, applying the interest operator and especially the correlator, and statistical pruning of the results. The array processor should be able to do all this nearly 100 times faster. The last five minutes was devoted to higher level tasks; maintaining the world model, path planning and generating graphical documentation of the program's thinking. Some steps in this section may be suitable for the array processor, but in any case we have found faster algorithms for much of it, for instance a shortest path in graph algorithm which makes maximum use of the sparsity of the distance matrix produced during the path planning.

We hope eventually to provide a manipulator on the Rover's topside, but there is no active work on this now. We chose the high steering flexibility of the current design partly to ease the requirements on a future arm. The weight and power needed can be reduced by using the mobility of the Rover to substitute for the shoulder joint of the arm. Such a strategy works best if the Rover body is given a full three degrees of freedom (X, Y and angle) in the plane of the floor. Conventional steering arrangements as in cars give only two degrees at any instant.

Rover Details

Three degrees of freedom of mobility are achieved by mounting the chassis on three independently steerable wheel assemblies (Figures 9-12). The control algorithm for this arrangement at every instant orients the wheels so that lines through their axles meet at a common point. Properly orchestrated this design permits unconstrained motion in any (2D) direction, and simultaneous independent control of the robot's rotation about its own vertical axis. An unexpected benefit of this agility is the availability of a "reducing gear" effect. By turning about the vertical axis while moving forward the robot derives a mechanical advantage for its motors. For a given motor speed, the faster the Rover spins, the slower it travels forward, and the steeper the slope it can climb. (Visualization of this effect is left as an exercise for the reader.)

To permit low friction steering while the robot is stationary, each assembly has two parallel wheels connected by a differential gear. The drive shaft of the differential goes straight up into the body of the robot. A concentric hollow shaft around this one connects to the housing of the differential (Figure 11). Turning the

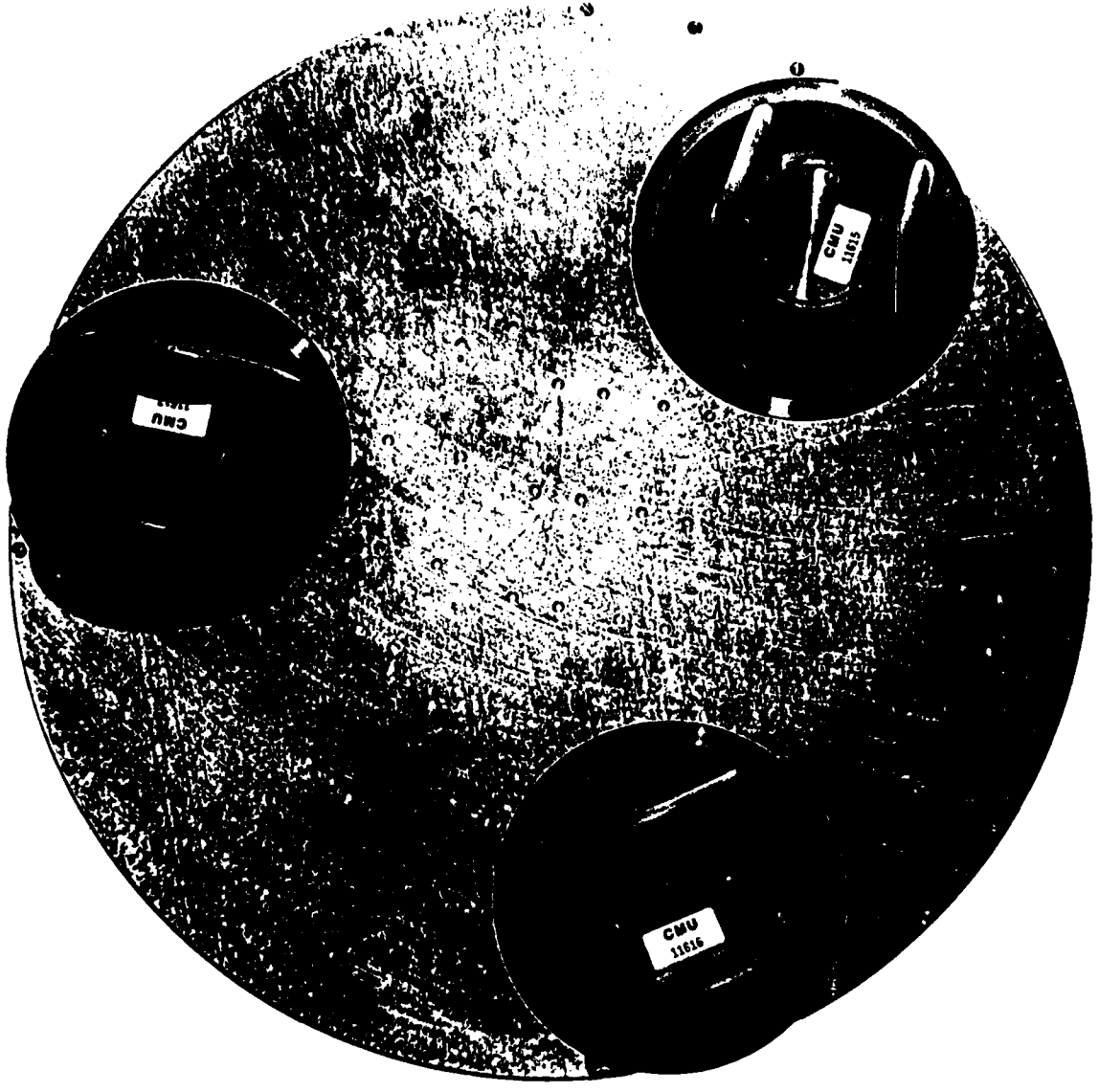


Figure 9: The Rover Wheelbase

The steering angle and drive of each wheel pair is individually controlled. The rover's trajectory will be an arc about any point in the floor plane if lines through the axles of all three wheels intersect at the center of that arc.

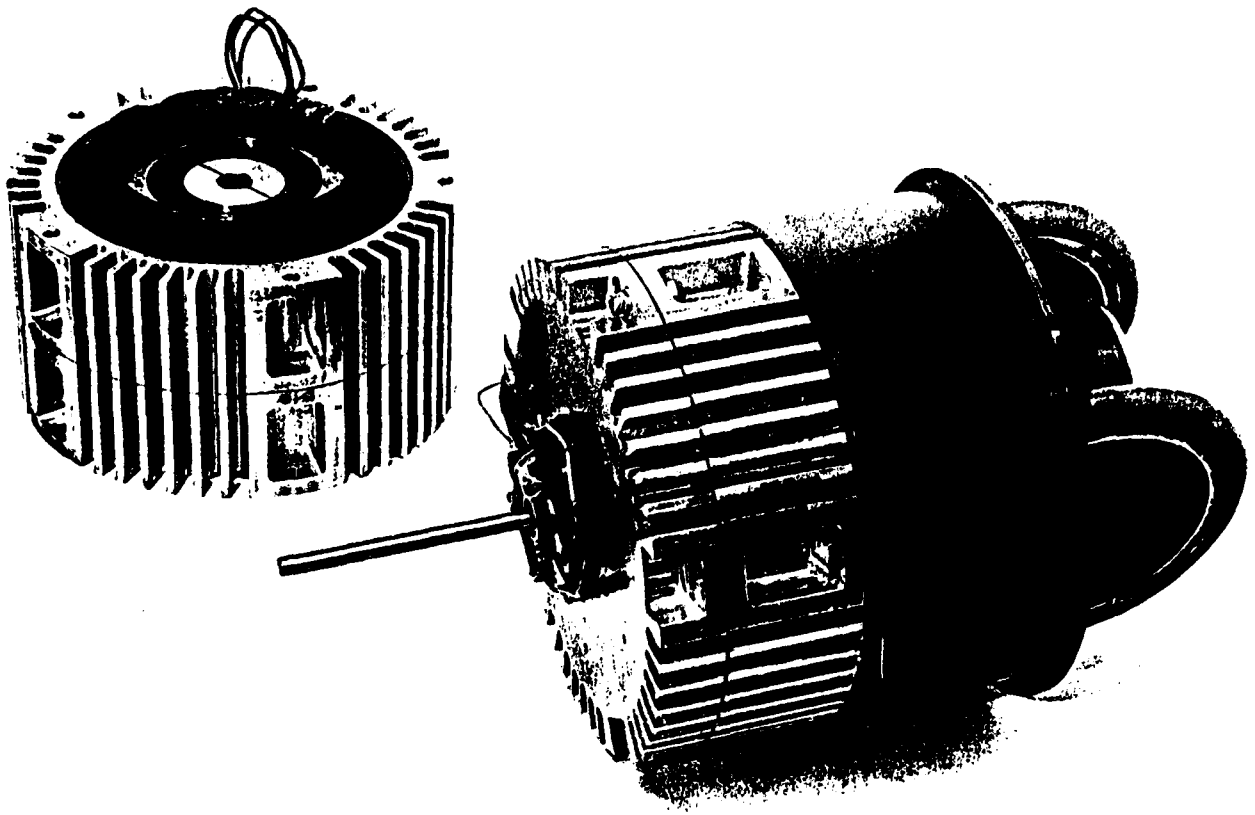


Figure 10: The Rover Wheel Assembly

The steering motor is shown attached to the wheel assembly, part of the drive motor is shown detached.

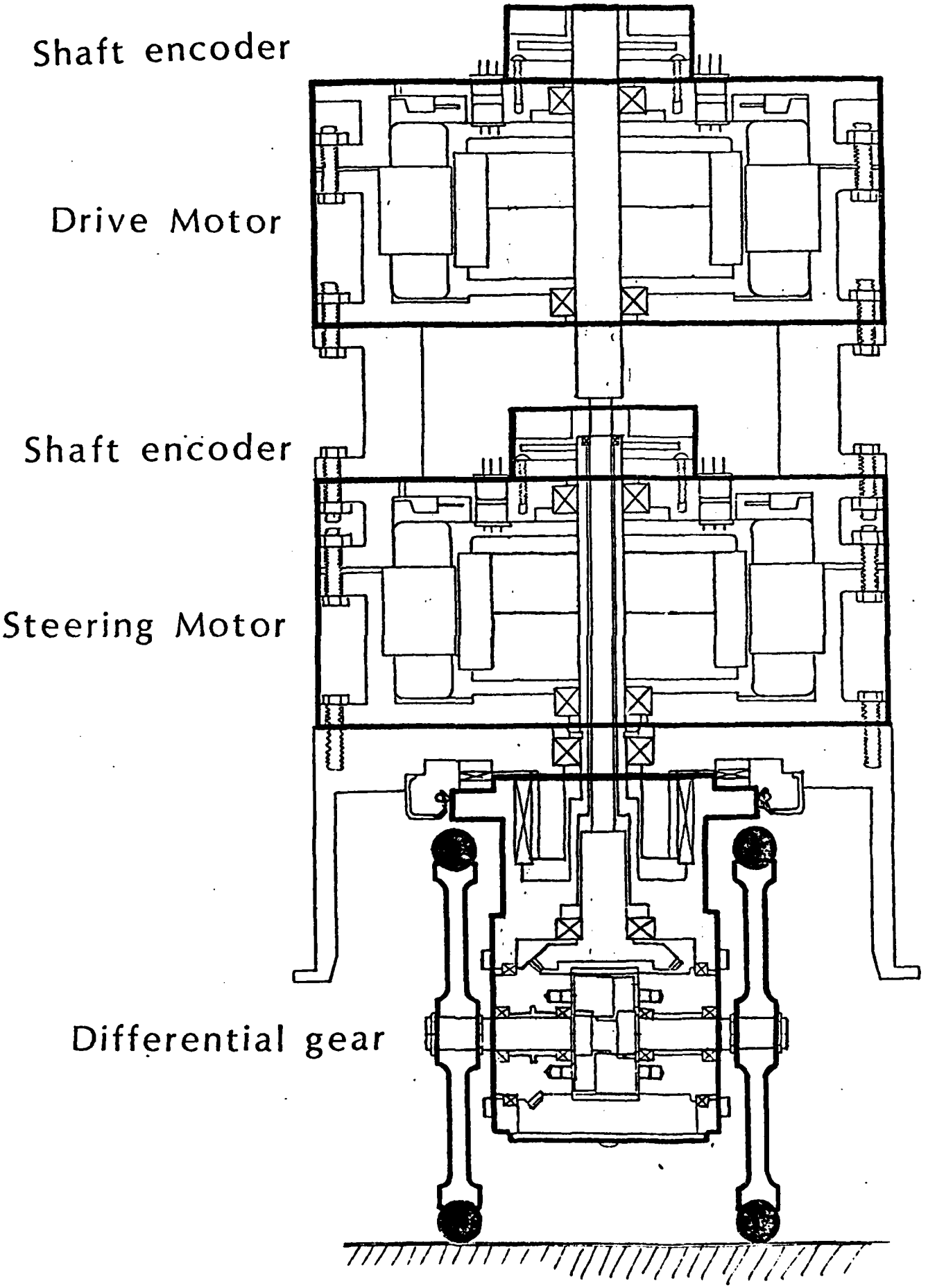
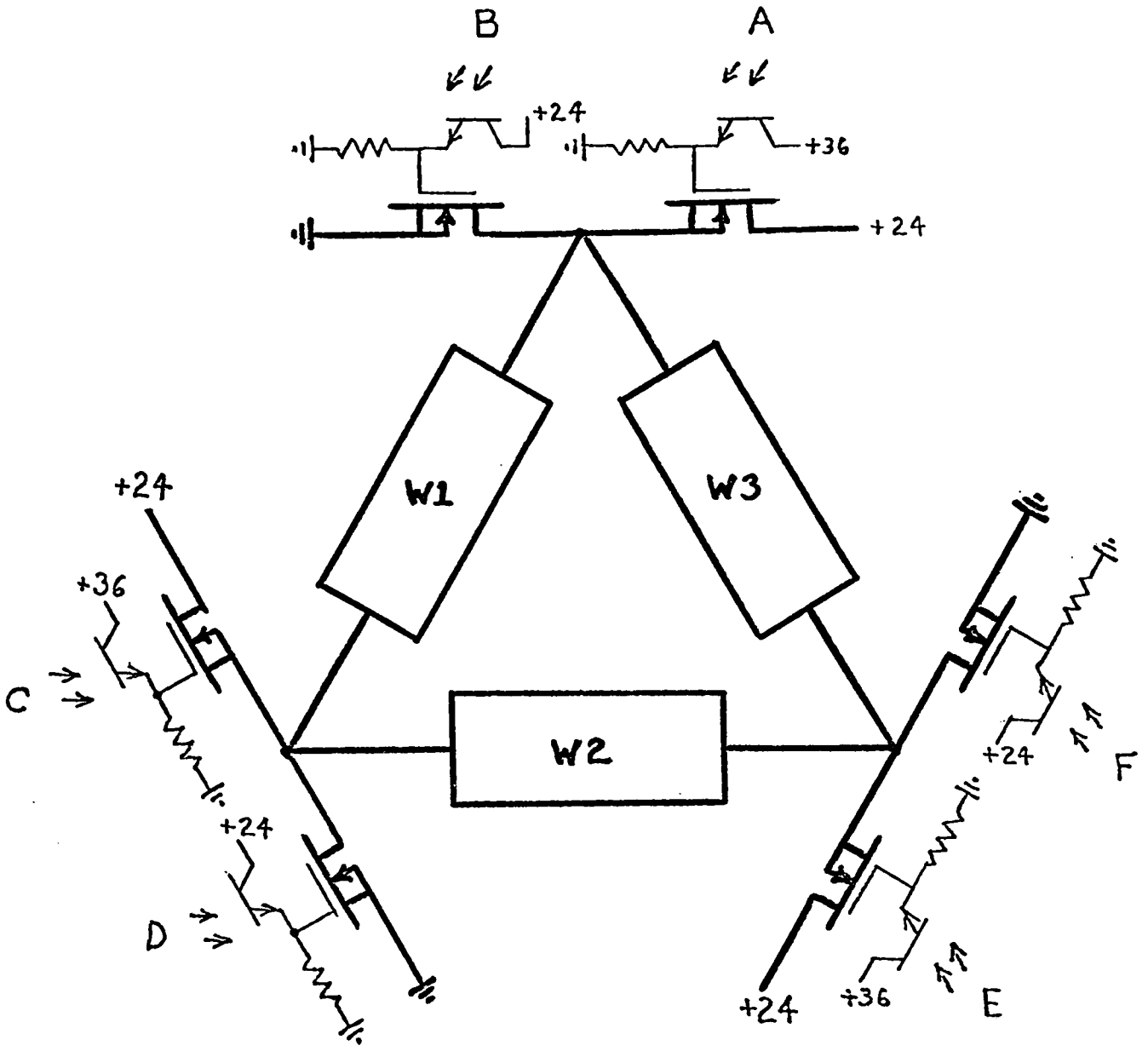


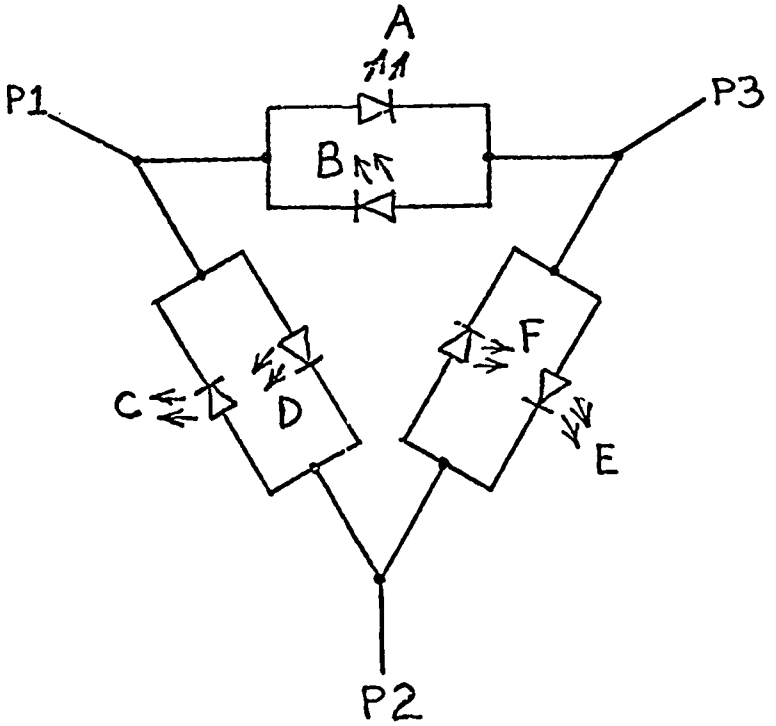
Figure 11: Diagram of the Rover Wheel Assembly

Figure 12: Rover Brushless Motor Drive Circuitry

Each Samarium-Cobalt brushless motor in the wheel assemblies is sequenced and servoed by its own microprocessor. A processor generates three bipolar logic signals *P1, P2* and *P3* which control the currents through the three phase motor windings *W1, W2* and *W3* through the IR light emitting diode/phototransistor optical links A through F shown. Each phototransistor controls a power field effect transistor which switches power to the windings. The circuitry in the upper, optically isolated, portion of the diagram is contained withing its motor's housing, on an annular circuit board, using the housing as heat sink. The LEDs poke through holes in the cover. Motor direction is controlled by sequencing order; torque is adjusted by pulse width modulation.

FIG 12





inner shaft causes the wheels to roll forwards or backwards, turning the outer one steers the assembly, with the two wheels rolling in a little circle. The assemblies were manufactured for us by Summit Gear Corp.

Each shaft is connected to a motor and a 4000 count/revolution optical shaft encoder (Datametrics K3). The two motors and two encoders are stacked pancake fashion on the wheel assembly, speared by the shafts. There are no gears except for the ones in the differential. (Figure 11 shows a schematic cross section of a complete motor/wheel-assembly structure, Figure 10 a partially assembled stack in the flesh).

The motors are brushless with samarium-cobalt permanent magnet rotors and three-phase windings (Inland Motors BM-3201). With the high energy magnet material, this design has better performance when the coils are properly sequenced than a conventional rotating coil motor. The coils for each are energized by six power MOSFETs (Motorola MTP1224) mounted in the motor casing and switched by six opto-isolators (to protect the controlling computers from switching noise) whose LEDs are connected in bidirectional pairs in a delta configuration, and lit by three logic signals connected to the vertices of the delta (Figure 12).

The motor sequencing signals come directly from onboard microprocessors, one for each motor. These are CMOS (Motorola MC146805 with Hitachi HM6116 RAMs) to keep power consumption low. Each processor pulse width modulates and phases its motor's windings, and observes its shaft encoder, to servo the motor to a desired motion (supplied by yet another processor, a Motorola 68000, the *Conductor* as a time parameterized function). Though the servo loop works in its present form, several approximations were necessary in this real-time task because of the limited arithmetic capability of the 6805. We will be replacing the 6805 with the forthcoming MC68008, a compact 8 bit bus version of the 68000.

The shaft encoder outputs and the torques from all the motors, as estimated by the motor processors, are monitored by another processor, *the Simulator*, a Motorola MC68000 (with all CMOS support circuitry the power requirement for our 32K 68000 is under one watt. The new high performance 74HC series CMOS allows operation at full 10MHz speed.), which maintains a dead-reckoned model of the robot's position from instant to instant. The results of this simulation (which represents the robot's best position estimate) are compared with the desired position, produced by another 68000, *the Controller*, in the previously introduced *the Conductor*, which orchestrates the individual motor processors. The Conductor adjusts the rates and positions of the individual motors in an attempt to bring the Simulator in line with requests from the Controller, in what amounts to a highly non-linear feedback loop.

Other onboard processors are:

Communication — A 68000 which maintains an error corrected and checked packet infrared link with a large controlling computer (a VAX 11/780 helped out by an ST-100 array processor and a custom high speed digitizer) which will do the heavy thinking. Programs run in the Controller are obtained over this link.

Sonar — A 6805 which controls a number of Polaroid sonar ranging devices around the body of the Rover. These will be used to maintain a rough navigation and bump avoidance model. All measurements and control functions of this processor and the following ones are available (on request over a serial link) to the Controller.

Camera — A 6805 which controls the pan, tilt and slide motors of the onboard TV camera. The compact camera broadcasts its image on a small UHF or microwave transmitter. The signal is received remotely and the video signal captured by a high bandwidth digitizer system and then read by the remote VAX. There are tentative plans for a minimal vision system using a 68000 with about 256K of extra memory onboard the Rover, for small vision tasks

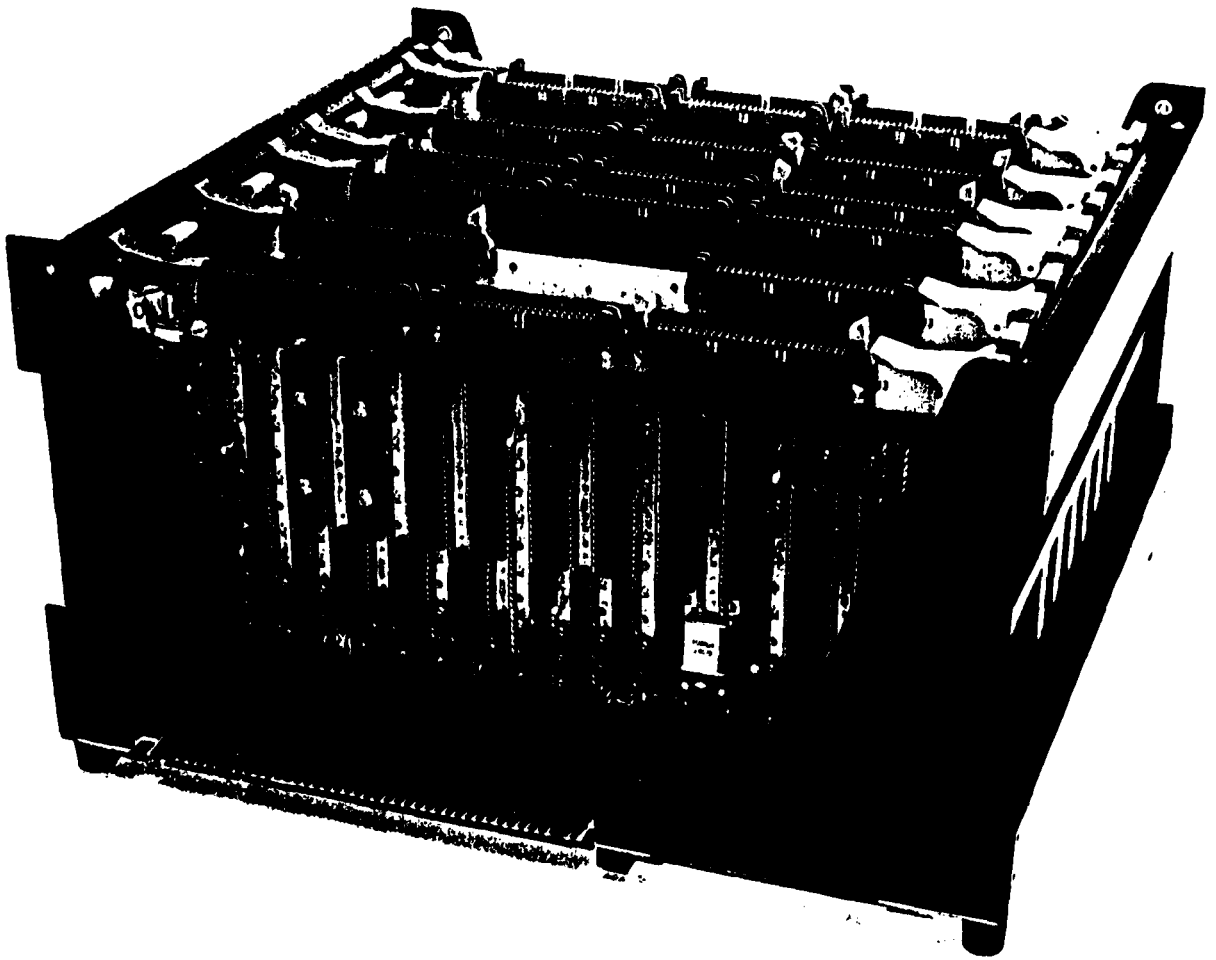


Figure 13: The Rover's Onboard Processors

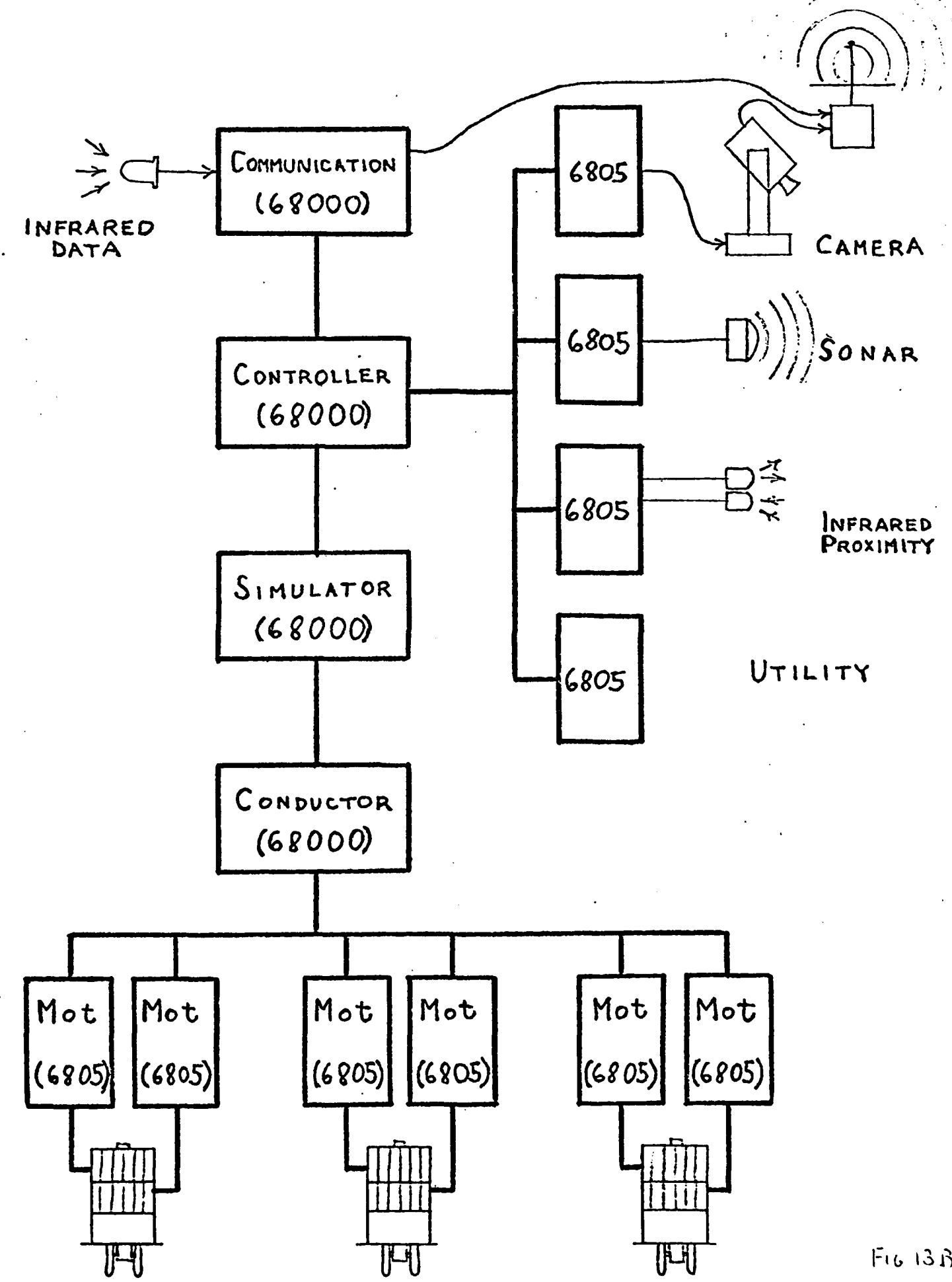

INFRARED DATA
COMMUNICATION (68000)
CONTROLLER (68000)
SIMULATOR (68000)
CONDUCTOR (68000)
6805
6805
6805
6805
CAMERA
SONAR
INFRARED PROXIMITY
UTILITY
Mot (6805)
Mot (6805)
Mot (6805)
Mot (6805)
Mot (6805)
Mot (6805)

FIG 13B

when the Rover is out of communication with the base system.

Proximity — A 6805 which monitors several short range modulated infrared proximity detectors which serve as a last line of defense against collision, and which sense any drop off in the floor, and contact switches.

Utility — A 6805 which senses conditions such as battery voltage and motor temperature, and which controls the power to non-essential but power hungry systems like the TV camera and transmitter.

Communication between processors is serial, via Harris CMOS UARTs, at a maximum speed of 256 kilobaud. The Conductor talks with the motor processors on a shared serial line and the Controller communicates with the Sonar, Camera, Proximity, Utility and any other peripheral processors by a similar method.

The processors live in a rack on the second storey of the robot structure (Figure 8, 13), between the motor and battery assembly (first floor) and the camera plane (penthouse). Figure 13 shows the initial interconnection.

The Rover is powered by six sealed lead-acid batteries (Globe gel-cell 12230) with a total capacity of 60 amp hours at 24 volts. The motors are powered directly from these, the rest of the circuitry derives its power indirectly from them through switching DC/DC converters (Kepco RMD-24-A-24 and Semiconductor Circuits U717262). Each 6805 processor draws about one eighth of a watt, each 68000 board only one watt.

Physically the robot is a meter tall and 55 cm in diameter. It weighs 90 kilograms. The maximum acceleration is one quarter g, and the top speed is 10 kilometers/hour. With appropriate onboard programming the motions should be very smooth. The great steering flexibility will permit simulation of other steering systems such as those of cars, tanks and boats and other robots by changes in programming.

Progress

As of this writing (January 1983) the robot's major mechanical and electronic structures are complete, mostly tested, and being assembled. An instability in the servo control algorithms which had held us up for several months has finally been solved, and we expect baby's first "steps" early in 1983. The digitizer unit which will receive and generate video and other data for the robot is under construction. Its specifications include four channels each with two megabytes of memory and two ports able to transfer data at 100 megabytes per second.

Promises

The high level control system has become very interesting. Initially we had imagined a language in which to write scripts for the onboard Controller similar to the AL manipulator language developed at Stanford [6], from which the commercial languages VAL at Unimation [15] and the more sophisticated AML [14] at IBM were derived. Paper attempts at defining the structures and primitives required for the mobile application revealed that the essentially linear control structure of these state-of-the-art arm languages was inadequate for a rover. The essential difference is that a rover, in its wanderings, is regularly "surprised" by events it cannot anticipate, but with which it must deal. This requires that routines for responding with various situations can be activated in arbitrary order, and run concurrently.

We briefly examined a production system, as used in many "expert systems", as the framework for the requisite real time concurrency, but have now settled on a structure similar to that developed for the CMU

Hearsay II speech understanding project [5]. Independent processes will communicate via messages posted on a commonly accessible data structure we call a *blackboard.* The individual processes, some of which will run under control of a spare real time operating system on one or more of the onboard 68000s, others of which will exist at the other end of a radio link on the VAX, change their relative priority as a consequence of relevant messages on the blackboard. For instance, a note from several touch sensors signalling a collision is taken as a cue by the avoidance routine to increase its running rate, and to post messages which trigger the motor co-ordinating routines to begin evasive actions. We plan to implement the multiple processes required for this task on each of several of the onboard 68000's with the aid of a compact (4K), efficient real time operating system kernel called VRTX available from Hunter & Ready. A more detailed description of the state of this work may be found in [4].

Other interesting preliminary thinking has resulted in a scheme by which a very simple arm with only three actuators will enable the robot, making heavy use of its great steering flexibility, to enter and leave through a closed standard office door (Figure 14).

Stepping into a more speculative realm, we are considering approaches to model based vision [2] which would permit recognition of certain classes of objects seen by the robot. Discussions with the distributed sensor net crew here at CMU [13] has raised the possibility of equipping the robot with ears, so it could passively localize sound, and thus perhaps respond correctly, both semantically and geometrically, to a spoken command like "Come here!" (using, in addition, speech understanding technology also developed at CMU [16]).

We are also toying with the idea of a phased array sonar with about 100 transducers operating at 50 KHz which, in conjunction with the high speed analog conversion device mentioned above and the array processor, would be able to produce a modest resolution depth map (and additional information) of a full hemisphere in about one second, by sending out a single spherical pulse, then digitally combining the returned echoes from the individual transducers with different delay patterns to synthesize narrow receiving beams.

Philosophy

It is my view that developing a responsive mobile entity is the surest way to approach the problem of general intelligence in machines.

Though computers have been programmed to do creditable jobs in many *intellectual* domains, competent performance in *instinctive* domains like perception and common sense reasoning is still elusive. I think this is because the instinctive skills are fundamentally much harder. While human beings learned most of the intellectual skills over a few thousand years, the instinctive skills were genetically honed for hundreds of millions of years, and are associated with large, apparently efficiently organized, fractions of our brain; vision, for example, is done by a specialized 10% of our neurons. Many animals share our instinctive skills, and their evolutionary record provides clues about the conditions that foster development of such skills. A universal feature that most impresses me in this context is that *all animals that evolved perceptual and behavioral competence comparable to that of humans first adopted a mobile way of life.*

This is perhaps a moot point in the case of the vertebrates, which share so much of human evolutionary history, but it is dramatically confirmed among the invertebrates. Most molluscs are sessile shellfish whose behavior is governed by a nervous system of a few hundred neurons. Octopus and squid are molluscs that abandoned life in the shell for one of mobility; as a consequence they developed imaging eyes, a large (annular!) brain, dexterous manipulators and an unmatched million channel color display on their surfaces. By contrast no sessile animal nor any plant shows any evidence of being even remotely this near to the human behavioral competence.

My conclusion is that *solving the day to day problems of developing a mobile organism steers one in the*

direction of general intelligence, while working on the problems of a fixed entity is more likely to result in very specialized solutions.

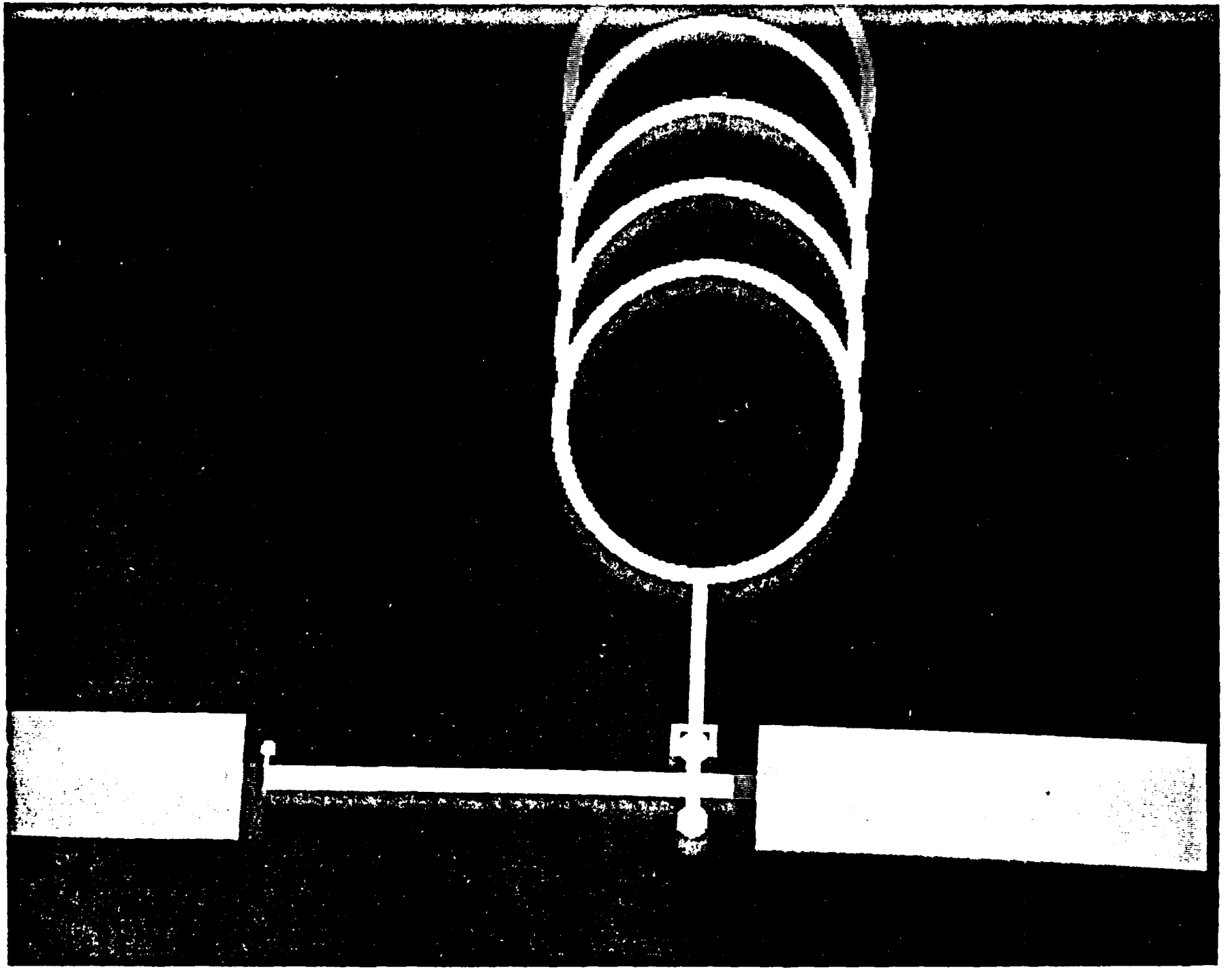
I believe our experience with the control language for the Rover vis a vis the languages adequate for a fixed arm, is a case in point. My experiences with computer vision during the Cart work re-inforce this opinion; constantly testing a program against fresh real-world data is nothing at all like optimizing a program to work well with a limited set of stored data. The variable and unpredictable world encountered by a rover applies much more selection pressure for generality and robustness than the much narrower and more repetitive stimuli experienced by a fixed machine. Mobile robotics may or may not be the fastest way to arrive at general human competence in machines, but I believe it is one of the surest roads.

Related Work

Other groups have come to similar conclusions, and have done sophisticated mobile robot work in past [12], [17]. The robotics group at Stanford has acquired a new, experimental, mobile robot from Unimation Inc., and plans research similar to ours [1]. This new Unimation rover [3] is very similar in size, shape and mechanical capabilities to the machine we are building. It achieves a full three degrees of freedom of floor-plane mobility by use of three novel "omnidirectional" wheels which, by virtue of rollers in place of tires, can freely move in a direction broadside to the wheel plane as well as performing the usual wheel motion under motor control.

Figure 14: The Rover Goes Through a Closed Door

Using a simple arm with only three powered actuators and two passive hinges, greatly helped by the wheel flexibility, the Rover deals with a self-closing office door. The door and knob are visually located, the Rover extends its arm, and approaches the door (a). The arm grasps the knob, twists it open, and the Rover backs up in an arc, partially opening the door (b). The Rover rolls around the door edge, while retaining its grasp on the knob; passive hinges in the arm bend in response (c). The Rover body now props open the door; the arm releases and retracts, and the Rover rolls along the front of the door (d). The Rover moves in an arc outward, allowing door to close behind it (e).



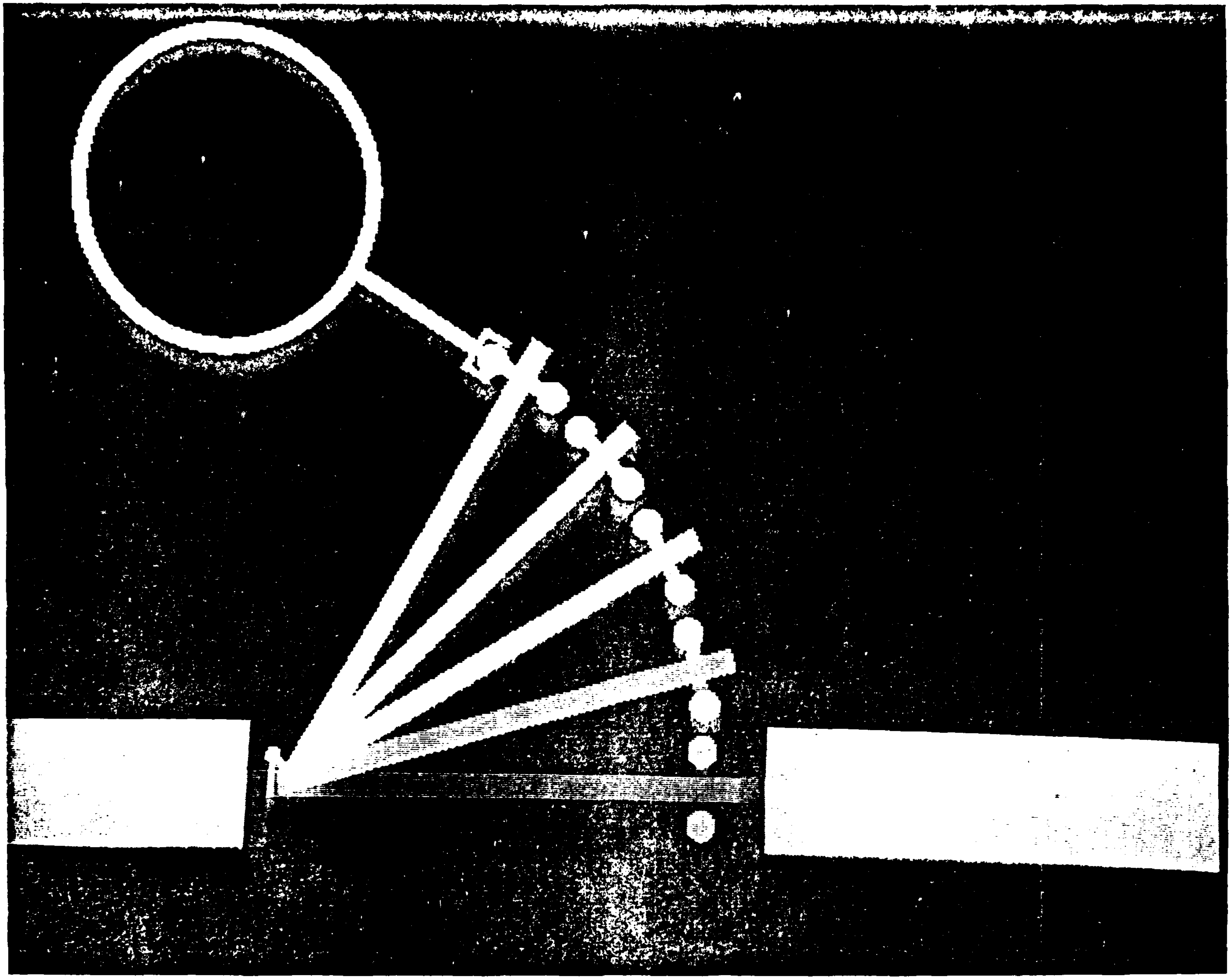
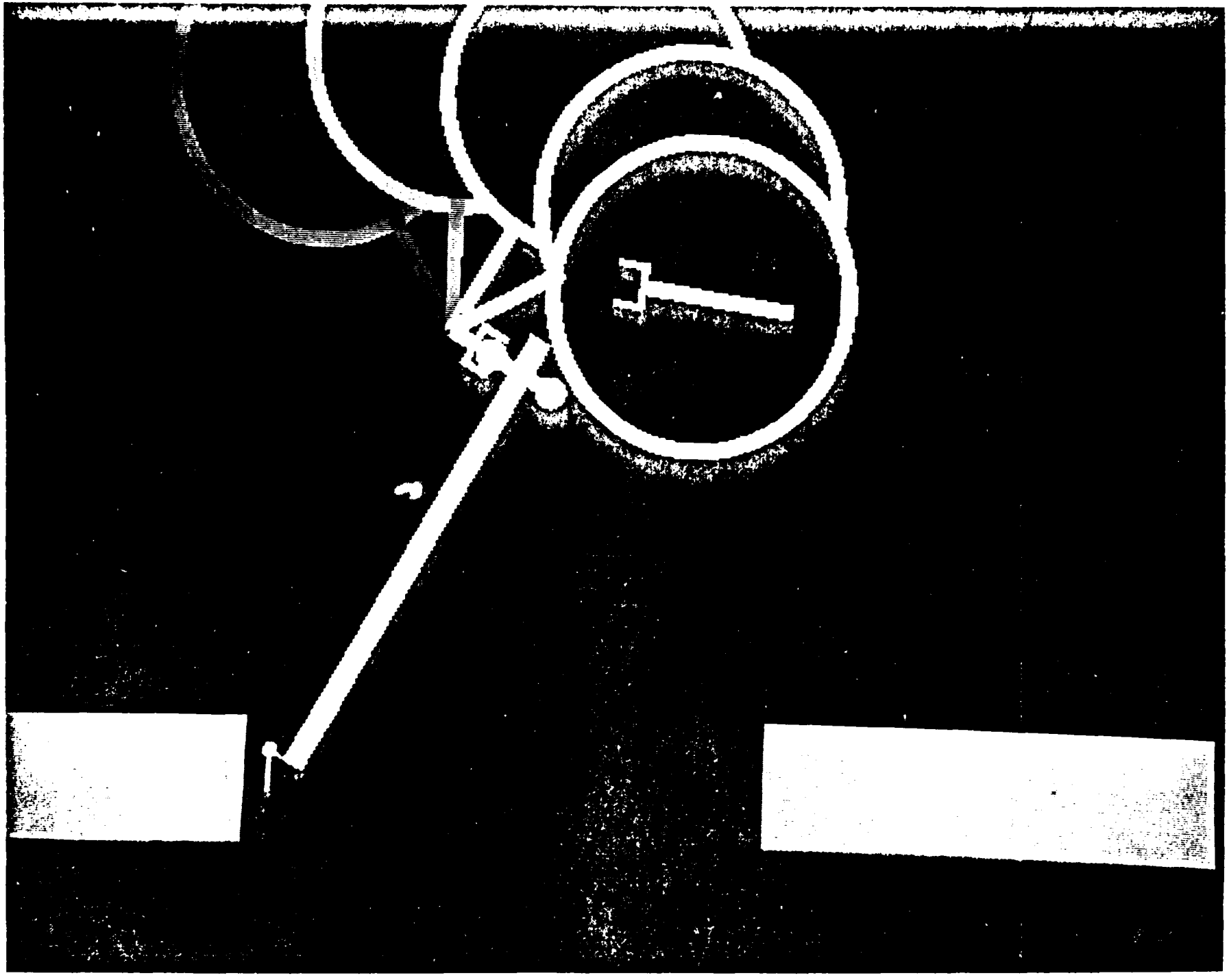


FIG. 14B



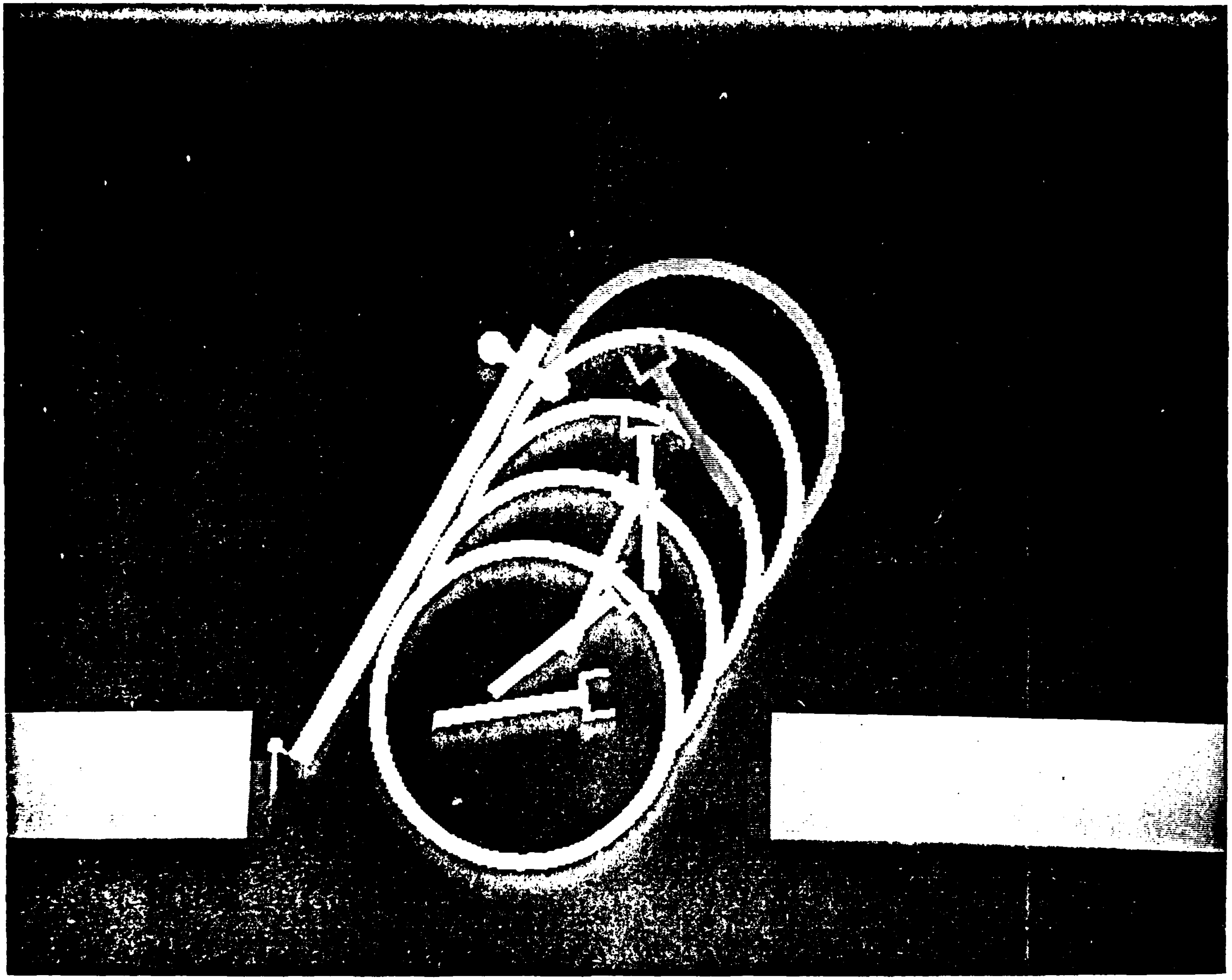


Figure D

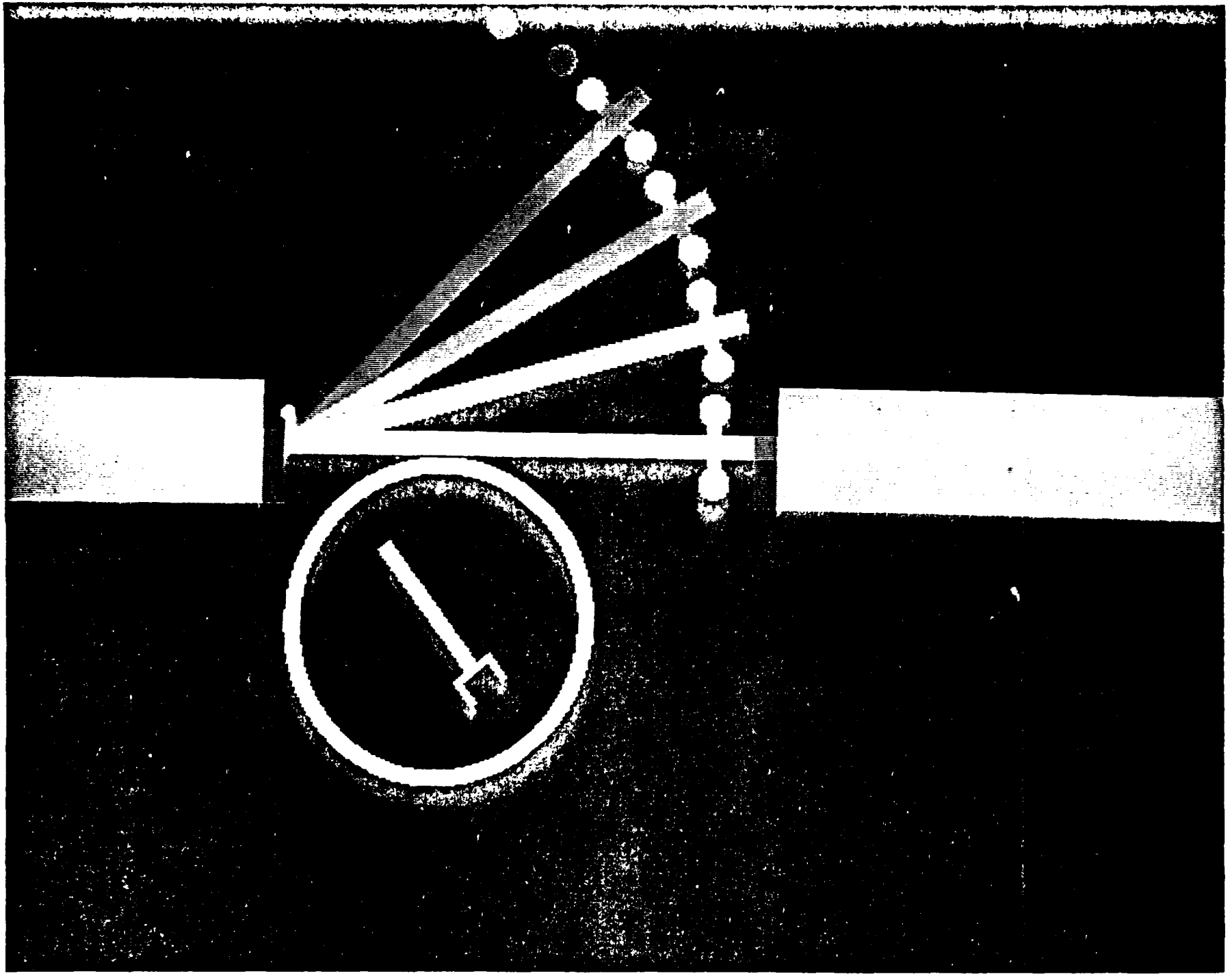


FIG 14 E

References

[1] Binford, T. O.
The Stanford mobile robot.
personal communication.
Stanford University Computer Science Dept., October 1982.

[2] Brooks, R. A.
Symbolic Reasoning Among 3-D Models and 2-D Images.
PhD thesis, Stanford University, June, 1981.

[3] Carlisle, B. and B. Shimano.
The Unimation mobile robot.
personal communication.
Unimation Inc., Mountain View, Ca., August 1981.

[4] Elfes, A. and S. N. Talukdar.
A Distributed Control System for the CMU Rover.
In *submitted to the 8th International Joint Conference on Artificial Intelligence, Karlsruhe, West Germany.* IJCAI, August, 1983.

[5] Erman, L. D. and V. R. Lesser.
The HEARSAY-II speech-understanding system: Integrating knowledge to resolve uncertainty.
Communications of the ACM 23(6), June, 1980.

[6] Goldman, R. and Shahid Mujtaba.
AL User's Manual, Third Edition.
Computer Science STAN-CS-81-889 (AIM-344), Stanford University, December, 1981.

[7] Lozano-Perez, T. and M. A. Wesley.
An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles.
Communications of the ACM 22(10):560-570, October, 1979.

[8] Moravec, Hans P.
Towards Automatic Visual Obstacle Avoidance.
In *Proceedings of the 5th International Joint Conference on Artificial Intelligence, MIT, Cambridge, Mass.*, pages 584. IJCAI, August, 1977.

[9] Moravec, Hans P.
Visual Mapping by a Robot Rover.
In *Proceedings of the 6th International Joint Conference on Artificial Intelligence, Tokyo, Japan*, pages 599-601. IJCAI, August, 1979.

[10] Moravec, Hans P.
Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover.
PhD thesis, Stanford University, September, 1980.
published as ***Robot Rover Visual Navigation*** by UMI Research Press, Ann Arbor, Michigan, 1981.

[11] Moravec, Hans P.
Rover Visual Obstacle Avoidance.
In *Proceedings of the 7th International Joint Conference on Artificial Intelligence, Vancouver, British Columbia*, pages 785-790. IJCAI, August, 1981.

[12] Raphael, B.
The Thinking Computer.
W. H. Freeman and Company, San Francisco, California, 1976.

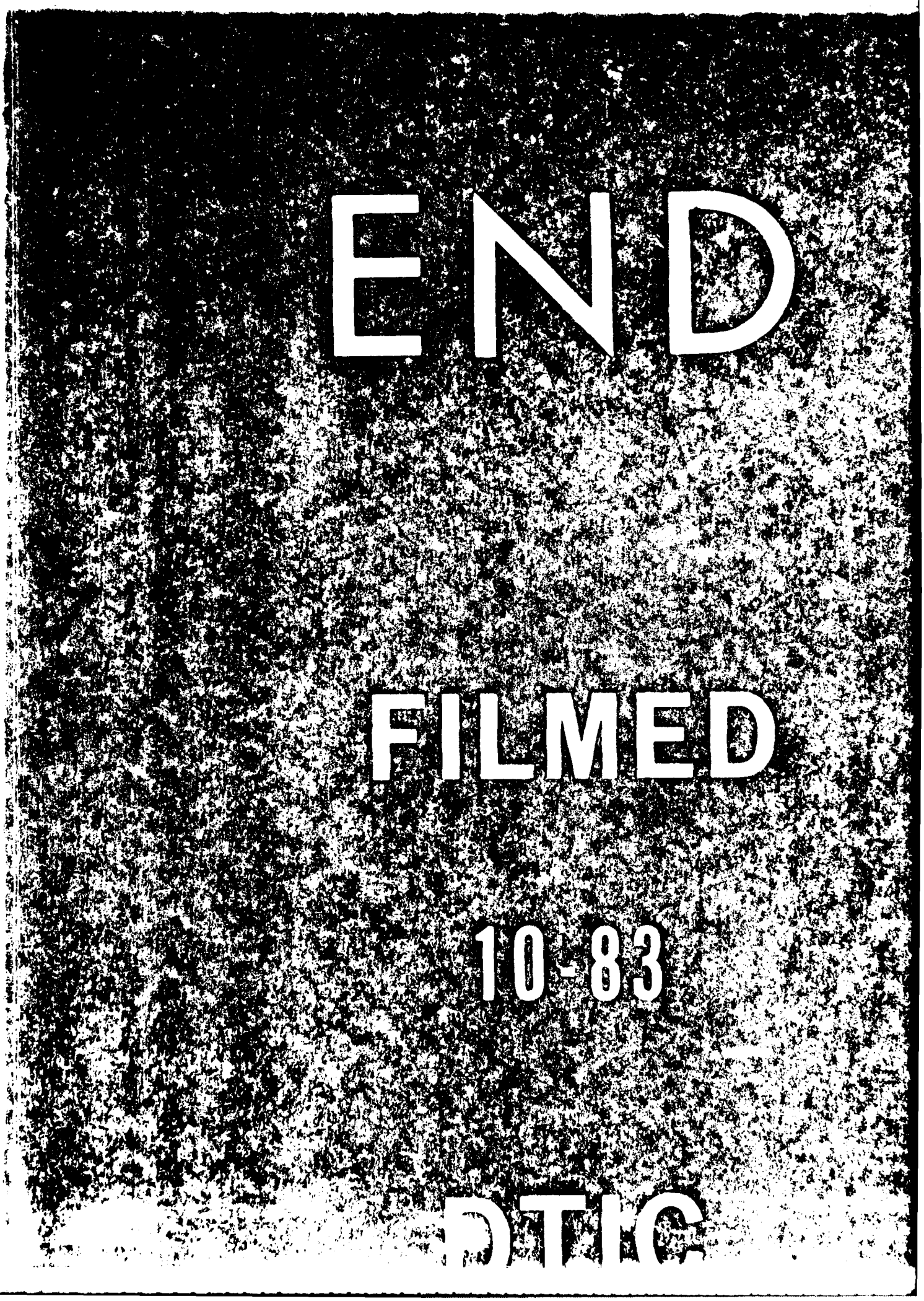
[13] Rashid, R. F. and G. G. Robertson.
Accent, A Communication Oriented Operating System Kernel.
Technical Report CS-81-123, CMU, October, 1981.

[14] Taylor R. H., P. D. Summers, and J. M. Meyer.
AML: A Manufacturing Language.
Research Report RC-9389, IBM, April, 1982.

[15] Unimation, Inc.
User's Guide to VAL, A Robot Programming and Control System, Version 11.
Technical Report, Unimation, Inc., February, 1979.

[16] Waibel, A., Yegnanarayana, B.
Comparative Study of Nonlinear Time Warping Techniques For Speech Understanding.
Technical Report 125, Carnegie-Mellon University Computer Science Department, 1981.

[17] Yakimovsky, Y. and R. Cunningham.
A System for Extracting Three-Dimensional Measurements from a Stereo Pair of TV Cameras.
Computer Graphics and Image Processing 7:195-210, 1978.


END
FILMED
10-83
DTIC